Explicit Control of Vector Field Based Shape Deformations

Wolfram von Funck
MPI Informatik
66123 Saarbruecken, Germany
wfunck@mpi-inf.mpg.de

Holger Theisel
Bielefeld University
33501 Bielefeld, Germany
theisel@techfak.uni-bielefeld.de

Hans-Peter Seidel
MPI Informatik
66123 Saarbruecken, Germany
hpseidel@mpi-inf.mpg.de

Abstract

Vector Field Based Shape Deformations (VFSD) have been introduced as an efficient method to deform shapes in a volume-preserving foldover-free manner. However, mainly simple implicitly defined shapes like spheres or cylinders have been explored as deformation tools by now. In contrast, boundary constraint modeling approaches enable the user to exactly define the support of the deformation on the surface. We present an approach to explicitly control VFSD: a scalar function together with two thresholds is placed directly on the shape to mark regions of full, zero, and blended deformation. The resulting deformation is volume-preserving and free of local self-intersections. In addition, the full deformation is steered by a 3D parametric curve and a parametric twisting function. This way our deformations appear to be a generalization of the boundary constraint modeling metaphor. We apply our approach in different scenarios. A parallelization of the computation on the GPU allows for editing high-resolution meshes at interactive speed.

1 Introduction

Shape deformations are a well-studied and often-addressed issue in Computer Graphics. A multitude of techniques has been developed in order to assist the user in various tasks like shape modeling, industrial design, physical simulation, animation and many more.

In the context of shape modeling, [23] introduced a space deformation technique based on vector fields. The idea of this approach, called *Vector Field Based Shape Deformations* (VFSD), is to construct a time-dependent divergence-free vector field $\mathbf{v}$ which defines the deformation. Meshes can be deformed by performing a path line integration of each vertex in $\mathbf{v}$ over a certain time. Due to the zero divergence of $\mathbf{v}$, the volume of the shape does not change during the deformation. Furthermore, the deformation will not produce self-intersections of the deformed shape, because path lines never intersect in space-time domain. These are desirable properties: volume-preserving deformations tend to look more plausible, whereas self-intersections are physically impossible and are not reversible using space deformation techniques.

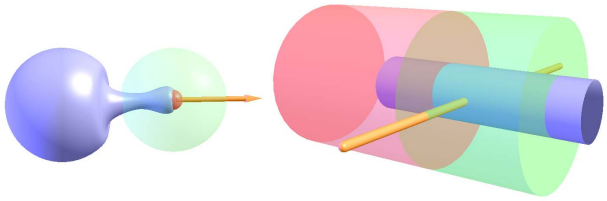

Figure 1. Using implicitly defined tools, it is difficult to exactly control the influence of the deformation (images from [23]).

However, the approach presented in [23] has a severe drawback: unlike in most modern shape editing frameworks, it is not possible to define exactly which parts of the shape surface are to be deformed. Instead, [23] use trivariate scalar functions which implicitly define simple tools like spheres or cylinders which can be used to deform the shape. Figure 1 illustrates this. In many situations, it is difficult or even impossible to achieve the desired deformation using such tools.

In this paper, we present an approach based on VFSD which eliminates the above-mentioned drawback by defining the regions of deformations directly on the shape and to

define the full deformation by a 3D parametric curve and a scalar twisting function. This approach is mainly motivated by boundary constraint modeling approaches. They are widely used in recent shape editing frameworks: by placing boundary constraints directly on the surface, the user can exactly define which parts of the surface are deformed by what amount. In fact, from the user's point of view our approach can be considered as a generalization of boundary constraint modeling: by choosing appropriate parameters, our method gives a similar look-and-feel as boundary constraint modeling approaches. However, we also show that our method can handle deformations which are impossible to obtain by boundary constraint modeling.

The rest of the paper is organized as follows: Section 2 gives an overview of previous shape deformation approaches and reviews the VFSD method. Section 3 describes our approach from the user's point of view. Section 4 shows the underlying vector field construction. Section 5 gives details about our GPU-based implementation. Section 6 describes different applications, among them a scenario which - from the users's point of view - is similar to boundary constraint modeling. Section 7 evaluates our method, while conclusions are drawn in section 8.

2 Related Work

Out of the large pool of existing shape deformation approaches, we want to give an overview of the most relevant ones, with respect to our work. These include boundary constraint modeling, handle-controlled deformations, volume-preserving deformations and foldover-free deformations. Finally, we review the VFSD method.

Most boundary constraint modeling techniques are surface-based and compute the deformation as the solution of a curvature energy minimization problem with respect to given boundary constraints. By modifying the boundary constraints, the user can precisely control the deformation. Multiresolution methods [14, 7] allow for fast computations and preservation of small-scale features. Recent approaches solve the Laplace/Poisson equations [1, 15, 22, 24, 16, 25] and emphasize correct deformation of local detail. These methods are based on the solution of large sparse linear systems and require a certain sampling quality of the deformed mesh. In order to increase efficiency and robustness, [8] introduced a boundary constraint modeling technique based on space deformations. Besides the ability to deform non-polygonal surfaces like point-based models, the approach allows for interactive deformations of highly complex geometry by shifting the computation to the GPU.

Many techniques use the concept of a control handle: the user selects some part of the surface and deforms it in order to control the deformation of the complete shape. Prominent representatives of this approach are Wires [21], implictly defined occluders [5], shape modeling with point-sampled geometry [19] and Twister [17]. These methods are efficient because they don't rely on linear system solvers. However, the deformed surface is not optimized with respect to curvature energy.

In order to produce natural, physically plausible deformations, several techniques have been designed that preserve the volume of a shape under deformation. While [26] uses a volumetric graph Laplacian to preserve volumetric details, other approaches explicitly try to keep the volume constant, either globally [11, 9, 20, 4, 13] or locally [6]. The Swirling Sweepers method [2] preserves volume implicitly by utilizing basic volume-preserving space deformations.

Self-intersecting surfaces are undesirable because of two reasons: they are physically impossible and the cannot be reversed by space deformations. A number of approaches dealing with this issue exist [18, 10, 6, 3, 2].

Being based on scalar and vector fields, the shape deformation technique in [12] is also related to our work, but takes a totally different approach.

Vector Field Based Shape Deformations

In [23], C^1 continuous time-dependent 3D divergence-free vector fields $\mathbf{v}(\mathbf{x}, t)$ are constructed in order to deform the meshes via pathline integration of their vertices. The resulting space deformation is volume-preserving, selfintersection-free, independent of the shape representation and requires no precomputation. For the sake of completeness, we review the method here.

The vector field $\mathbf{v}$ is computed as

$$\mathbf{v}(\mathbf{x}, t) = \nabla p(\mathbf{x}, t) \times \nabla q(\mathbf{x}, t) \tag{1}$$

where p, q are piecewise C^2 continuous time-dependent scalar fields and ∇ describes the (spatial) gradient. The regions of deformation are defined by a time-dependent scalar field $r(\mathbf{x}, t)$ and two constant thresholds $r_i < r_o$: if a point $\mathbf{x}$ is in the inner region at the time t (i.e., if $r(\mathbf{x}, t) < r_i$), $\mathbf{x}$ undergoes a full deformation at the time t; if $\mathbf{x}$ is in the outer region (i.e. $r_o \leq r(\mathbf{x}, t)$), $\mathbf{x}$ remains undeformed, and for $\mathbf{x}$ in the intermediate region (i.e. $r_i \leq r(\mathbf{x}, t) < r_o$), the deformation of $\mathbf{x}$ is obtained by a blending process. This way we get

$$p(\mathbf{x}, t) = \begin{cases} e(\mathbf{x}, t) & \text{if} \quad r(\mathbf{x}, t) < r_i \\ (1 - b(\mathbf{x}, t)) \cdot e(\mathbf{x}, t) \; + \; b(\mathbf{x}, t) \cdot 0 & \\ & \text{if} \quad r_i \leq r(\mathbf{x}, t) < r_o \\ 0 & \text{if} \quad r_o \leq r(\mathbf{x}, t) \end{cases} \tag{2}$$

$$q(\mathbf{x}, t) = \begin{cases} f(\mathbf{x}, t) & \text{if} \quad r(\mathbf{x}, t) < r_i \\ (1 - b(\mathbf{x}, t)) \cdot f(\mathbf{x}, t) \; + \; b(\mathbf{x}, t) \cdot 0 & \\ & \text{if} \quad r_i \leq r(\mathbf{x}, t) < r_o \\ 0 & \text{if} \quad r_o \leq r(\mathbf{x}, t) \end{cases} \tag{3}$$

where e, f are simple time-dependent analytic (e.g., linear or quadratic) scalar fields, and the blending function b is defined as

$$b(\mathbf{x},t) = \sum_{i=0}^{4} w_i \; B_i^4 \left(\frac{r(\mathbf{x},t) - r_i}{r_o - r_i} \right) \qquad (4)$$

where B_i^4 describe the well-known Bernstein polynomials and $(w_0, ..., w_4) = (0, 0, 0, 1, 1)$. Then, different choices of e, f, r give different modeling metaphors. For instance, a translation of the inner region along a (normalized) vector $\mathbf{t}$ can be obtained by e, f as linear fields with $\nabla e \cdot \nabla f = \nabla e \cdot \mathbf{t} = \nabla f \cdot \mathbf{t} = 0$ and $(\nabla e)^2 = (\nabla f)^2 = 1$. To get a rotation around an axis defined by a point $\mathbf{z}$ and a vector $\mathbf{t}$, we choose

$$e(\mathbf{x},t) = \mathbf{t} \cdot (\mathbf{x} - \mathbf{z}) \quad , \quad f(\mathbf{x},t) = (\mathbf{t} \times (\mathbf{x} - \mathbf{z}))^2. \quad (5)$$

In [23], r has been chosen only implicitly to define simple regions (e.g., spheres, cylinders) as regions of deformation.

3 Our approach from the user's point of view

In this section we give a user's oriented view to our approach. The main difference to [23] is that we define the regions of deformation directly on the shape instead of implicitly by a scalar field. Given a shape (here defined as triangular mesh), the user defines a continuous scalar function $s(\mathbf{x})$ on the shape. Together with two thresholds $s_i < s_o$, s defines three regions of deformation on the shape: a vertex $\mathbf{x}$ in the inner region (i.e., $s(\mathbf{x}) \leq s_i$) undergoes a full deformation, a vertex $\mathbf{x}$ in the outer region (i.e., $s_o \leq s(\mathbf{x})$) remains undeformed, whereas the deformation in the intermediate region (i.e. $s_i < s(\mathbf{x}) < s_o$) is obtained by a blending approach. Without loss of generality, we use $s_i = 0$ and $s_o = 1$ throughout this paper.

Furthermore we note that in the inner and the outer region, s does not contribute to the computation of the deformation. Therefore we can safely set $s = 0$ in the inner region and $s = 1$ in the outer region. We achieve a continuous scalar function $s \in [0, 1]$ on the shape which defines for every vertex $\mathbf{x}$:

$$\begin{aligned} s = 0 \quad &\rightarrow \quad \text{full deformation} \\ 0 < s < 1 \quad &\rightarrow \quad \text{blended deformation} \\ s = 1 \quad &\rightarrow \quad \text{no deformation.} \end{aligned}$$

Figure 2 illustrates the scalar field s on a shape by color coding: blue means $s = 0$, red means $s = 1$, while the intermediate color values are smoothly color interpolated. In addition to this color coding, the separation curves between the different regions are highlighted. Also, in this and the following images, as well as the accompanying video, the small white sphere on the curve visualizes the current integration time, while the dark sphere represents the target time for the integration. The target time can be interactively moved by the user.

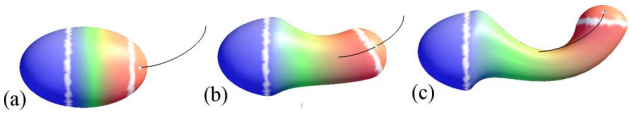

Figure 2. (a) The deformation is defined by two closed polygons on the shape and a parametric curve $\mathbf{c}(t)$; (b) path line integration at $t = 1/2$; (c) path line integration at $t = 1$ is the desired deformation.

During the deformation (i.e., the path line integration of the vertices), the location of the vertices change. During this process we keep $s(\mathbf{x})$ constant to the originally assigned values unless s is recomputed on the user's request at a certain stage of the deformation.

To define the full deformation (i.e., the deformation which vertices undergo in the inner region) itself, most modern approaches define a handle which is interactively placed at its destination point and orientation. Due to its nature, our approach is able to consider not only the end point of the deformation but also the way it takes. Therefore, we define the deformation by a parametric curve $\mathbf{c}(t)$, $t \in [0, 1]$, where $\mathbf{c}(0)$ lies on the inner region of the shape. Note that $\mathbf{c}$ can be constructed in two ways: either by explicitly defining the curve (e.g., as B-spline curve), or by interactively moving $\mathbf{c}(0)$. Figure 2 explains an example.

In addition, a twisting effect during the deformation can be defined by defining a continuous scalar function $\alpha(t)$, $t \in [0, 1]$. It describes the twisting angle during the deformation along $\mathbf{c}(t)$: during the complete deformation (i.e., the path line integration from $t = 0$ to 1), a twisting by the angle $\alpha(1) - \alpha(0)$ is carried out. If $\alpha(t)$ =const, no twisting is involved into the deformation.

4 Constructing the vector field

In this section we describe how to use VFSD with a control of the deformation as described in section 3. It turns out that we can rely on the vector field construction described in (1)–(4). In fact, we only have to modify the definition of e, f and r to get the desired control of the deformation. The choice of r is responsible to control the regions of deformation on the surface, while e, f describe the deformation in the inner region.

4.1 Constructing r

The function $r(\mathbf{x},t)$ together with r_i, r_o define the region of the deformation. Conceptually, r has to be defined as a time-dependent volumetric function, since both r and ∇r contribute to the definition of $\mathbf{v}$ by (1)–(4). However, r and ∇r are only evaluated at the surface of the shape, i.e. at the shape vertices. We use this fact to estimate r and ∇r at each vertex $\mathbf{x}$ out of the scalar field s which is only defined on the shape. In fact, we set $r(\mathbf{x},t) = s(\mathbf{x})$ for each vertex. To estimate $\nabla r(\mathbf{x},t)$, we consider s at $\mathbf{x}$ an all adjacent vertices in the 1-ring of $\mathbf{x}$. To do so, we apply a least-square fitting approach of the linear approximation of r in the neighborhood of $\mathbf{x}$: we solve

$$\left[\begin{array}{c} r(\mathbf{x},t) = s(\mathbf{x}) \quad , \quad \nabla r(\mathbf{x},t)\cdot\mathbf{n}(\mathbf{x},t) = 0 \\ \sum_{\mathbf{y}\in R_1(\mathbf{x})}(r(\mathbf{y},t)-s(\mathbf{y}))^2 \rightarrow \min \end{array}\right] \tag{6}$$

where $R_1(\mathbf{x})$ is the 1-ring of $\mathbf{x}$ and $\mathbf{n}(\mathbf{x},t)$ is the estimated shape normal at the vertex $\mathbf{x}$. This means that we assume a zero direction derivative of r in normal direction. Assuming r to be linearly approximated, (6) has a unique solution for r and ∇r. Finally we set $r_i = 0$ and $r_o = 1$ to get a complete estimation of r, r_i, r_o from the explicitly defined s.

4.2 Constructing e, f

The scalar fields e, f are responsible for the definition of the deformation in the inner region. They have to be chosen such that the deformation in the inner region follows the curve $\mathbf{c}(t)$, and no distortions in the inner region are introduced during the deformation. To do so, we can choose e, f to define a rotation or a translation, and

$$\mathbf{v}(\mathbf{c}(t),t) = \dot{\mathbf{c}}(t). \tag{7}$$

Since $\mathbf{c}(0)$ is in the inner region of the deformation, this is equivalent to

$$\nabla e(\mathbf{c}(t),t) \times \nabla f(\mathbf{c}(t),t) = \dot{\mathbf{c}}(t). \tag{8}$$

In order to define a translation in the inner region, we define $\mathbf{v}(\mathbf{x},t) = \mathbf{v}(t)$ as a time-dependent constant field in the inner region. To do so, e, f are time-dependent linear scalar fields with

$$\begin{aligned} \nabla e(\mathbf{x},t)\cdot\dot{\mathbf{c}}(t) = \nabla f(\mathbf{x},t)\cdot\dot{\mathbf{c}}(t) = \\ \nabla e(\mathbf{x},t)\cdot\nabla f(\mathbf{x},t) = 0, \end{aligned} \tag{9}$$

$$\begin{aligned} \|\nabla e(\mathbf{x},t)\| = \|\nabla f(\mathbf{x},t)\| = \sqrt{\|\dot{\mathbf{c}}(t)\|}, \\ e(\mathbf{c}(t),t) = f(\mathbf{c}(t),t) = 0. \end{aligned} \tag{10}$$

Note that (10) gives a unique definition of e, f except for one degree of freedom: the direction of ∇e (or ∇f respectively) can be chosen arbitrary but perpendicular to $\dot{\mathbf{c}}(t)$.

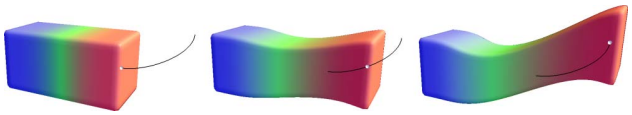

Figure 3. Local translation along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$: the inner region follows the curve but does not change its orientation.

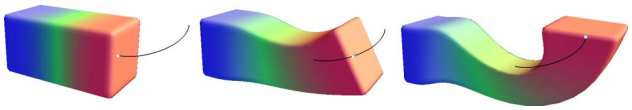

Figure 4. Local rotation along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$: the inner region follows the curve both in location and orientation.

However, it turns out that this degree of freedom does not have any influence on the definition of $\mathbf{v}$. Figure 3 illustrates a local translation of the inner region along the curve $\mathbf{c}(t)$. As we can see there, the inner region follows the curve $\mathbf{c}(t)$ but does not change its orientation.

In order to enable both the location and the orientation to follow $\mathbf{c}(t)$, we define the inner deformation as a rotation around a rotational axis perpendicular to the osculating plane of $\mathbf{c}$ and passing through the curvature center of $\mathbf{c}$. In fact, we compute the curvature center as

$$\mathbf{z}(t) = \mathbf{c} + \frac{\ddot{\mathbf{c}}(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})^2 - \dot{\mathbf{c}}(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})(\dot{\mathbf{c}}\cdot\ddot{\mathbf{c}})}{(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})(\ddot{\mathbf{c}}\cdot\ddot{\mathbf{c}}) - (\dot{\mathbf{c}}\cdot\ddot{\mathbf{c}})^2} \tag{11}$$

and the direction of the rotation axis as

$$\mathbf{t}(t) = (\dot{\mathbf{c}}\times\ddot{\mathbf{c}}). \tag{12}$$

Then $e(\mathbf{x},t), f(\mathbf{x},t)$ are defined as

$$e = \mathbf{t}\cdot(\mathbf{x}-\mathbf{z}) \quad , \quad f = (\mathbf{t}\times(\mathbf{x}-\mathbf{z}))^2 - (\mathbf{c}-\mathbf{z})^2. \tag{13}$$

It is straightforward exercise in algebra to show that (11), (12) and (13) give (8). Figure 4 illustrates a local rotation of the inner region: both its location and its orientation move along $\mathbf{c}(t)$.

In order to incorporate an additional twisting along the tangent of $\mathbf{c}$, we define an additional divergence-free vector field $\tilde{\mathbf{v}}$ out of the scalar fields $\tilde{e}, \tilde{f}, \tilde{r}$ similar to (1)–(4) by setting $\tilde{r} = r$ and $\tilde{e}, \tilde{f}$ defining a rotation around the axis $\mathbf{c}(t) + \lambda\,\dot{\mathbf{c}}(t)$. In fact, we set

$$\tilde{e}(\mathbf{x},t) = \frac{\dot{\alpha}(t)}{2\pi}(\mathbf{x}-\mathbf{c}(t))\cdot\dot{\mathbf{c}}(t), \tilde{f}(\mathbf{x},t) = ((\mathbf{x}-\mathbf{c}(t))\times\dot{\mathbf{c}}(t))^2. \tag{14}$$

Then $\tilde{e}, \tilde{f}$ define a new vector field $\tilde{\mathbf{v}}$ which is simply added to $\mathbf{v}(\mathbf{x},t)$ for the path line integration. Figure 5 illustrates a twisting effect.

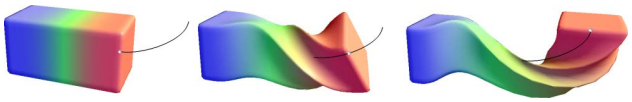

Figure 5. Local rotation and twisting along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$.

5 Implementation

In order to get interactive deformations for complex meshes, we shift the computation to the GPU. Here we use a General Purpose GPU Computing (GPGPU) approach: All necesseray data is stored in textures, and the computation is performed in a fragment shader by rendering a quad. When we want to deform a mesh, we first have to store the initial vertex positions together with the scalar field s in a texture, the *vertex texture*. Similarly, normals and gradients are stored in the *normal texture* and the *gradient texture*, respectively. Since the estimation of normals and gradients depends on the mesh connectivity, we also need to represent vertex connectivity as texture: We store up to eight indices of the neighbor vertices for each vertex in the *connectivity texture* (we only used meshes with a vertex valence smaller than nine). While the connectivity stays fixed throughout the deformation, vertex positions, normals and gradients are updated in each integration step. This is realized by four fragment shaders, which are called consecutively in each integration step.

Normal shader. In this shader, the normal of a vertex is computed as the normalized sum of the normals of the adjacent triangles. The necessary input is the vertex texture and the connectivity texture. The result is written to the normal texture.

Gradient shader. This shader estimates the gradient ∇r as described in Section 4.1. For this estimation, it uses the the vertex texture, the normal texture and the connectivity texture and renders the result to the gradient texture.

Smoothing shader. Due to small scale normal variations, the previously computed gradient is generally not smooth enough to give smooth deformations. To solve this problem, the smoothing shader performs a Laplacian smoothing on the estimated gradient: For a fixed number of steps, the gradient vector of each vertex is moved towards the mean gradient of its 1-ring neighborhood. This shader needs the normal texture, gradient texture and connectivity as input and overwrites the gradient texture.

Deformation shader. This shader performs the actual deformation after the gradient has been estimated. It computes a divergence-free vector field as described in Section 4 and performs one integration step of this field. In order to be robust, we use one fourth-order Runge Kutta step here. The input of this shader is the vertex texture and the gradient texture. The result is written to the vertex texture.

Using these shaders, the mesh can be deformed without readback from graphics memory, which would be a bottleneck. Only when we need to access the data (usually after the deformation), we read it from the respective textures.

6 Applications

In this section we introduce different choices of $s, \mathbf{c}, \alpha$ to obtain different application scenarios.

6.1 Geodesic level deformation

The main advantage of our approach over [23] is the fact that we can define the influence of the deformation directly on the surface, while [23] use implicitly defined scalar fields like the Euclidean distance for this purpose. Using a smooth approximation of the geodesic distance to a point on the surface, we can define the influence of the deformation more intuitively: The user simply selects a vertex on the shape, and the geodesic distance field $g(\mathbf{x})$ of this point is approximated on the surface. For this, we use Dijkstra's algorithm and smooth the scalar values afterwards using a Kernel with local support. That way, we make sure that the resulting deformation is smooth. Using two user-adjustable thresholds g_{min}, g_{max}, we define $s(\mathbf{x})$ as

$$s(\mathbf{x}) = \begin{cases} 1 - \frac{g(\mathbf{x}) - g_{min}}{g_{max} - g_{min}} & \text{if} \quad g_{min} < g(\mathbf{x}) < g_{max} \\ 1 & \text{if} \quad g(\mathbf{x}) \leq g_{min} \\ 0 & \text{if} \quad g_{max} \leq g(\mathbf{x}) \end{cases} \tag{15}$$

In Figure 7 the user has selected the middle finger of a hand shape by clicking on the tip of the finger. By adjusting the thresholds g_{min}, g_{max}, the regions of full and zero deformations have been set such that the deformation affects only this finger.

6.2 Emulation of boundary constraint modeling

In boundary constraint modeling, three regions of deformation are painted onto the surface. They correspond to the regions which are defined by the scalar field s. Note that s actually contains more information than used for boundary constrained modeling: in the intermediate region (i.e., at vertices $\mathbf{x}$ with $0 < s(\mathbf{x}) < 1$), the deformation depends on s. Therefore, in order to emulate boundary constraint modeling, the scalar field s in the intermediate region has to be chosen automatically. To do so, we compute the approximate geodesic distance $g_i(\mathbf{x})$ of a vertex $\mathbf{x}$ to the inner region on the shape, and we compute the approximate geodesic distance $g_o(\mathbf{x})$ of $\mathbf{x}$ to the outer region. Here we use the same smooth approximation as in Section 6.1. In

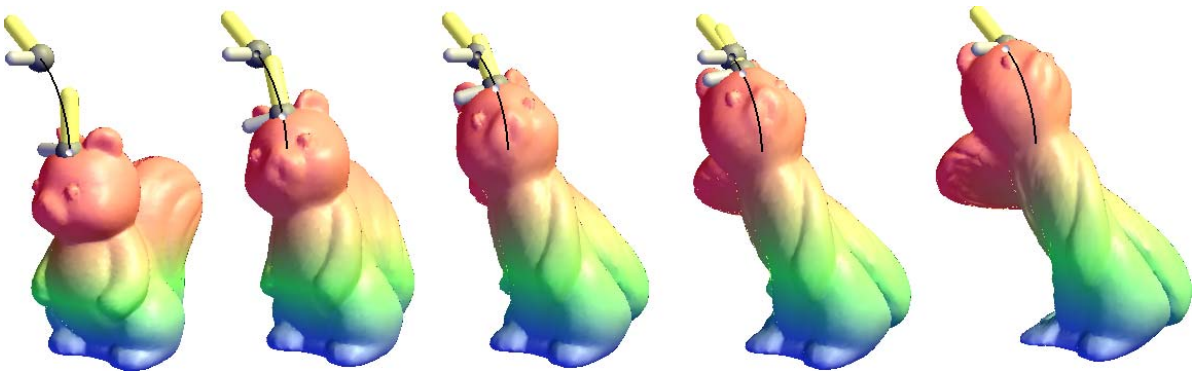

Figure 6. The squirrel model is deformed from start to target handle.

Figure 7. Using the geodesic distance field of a point on the tip of the middle finger, we can deform the finger without affecting other parts of the shape.

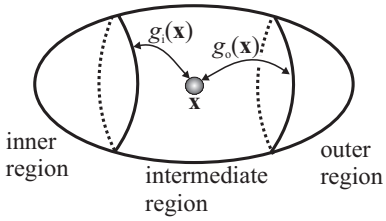

Figure 8. Computing $s(\mathrm{x})$ by approximating the geodesic distance between a vertex x and the inner/outer region.

fact, we compute and store the scalar fields g_i and g_o for every vertex in the inner region first by growing from the boundary between inner and intermediate, and intermediate and outer region respectively. Then we set

$$s(\mathbf{x}) = \frac{g_o(\mathbf{x})}{g_i(\mathbf{x}) + g_o(\mathbf{x})}\, 0 + \frac{g_i(\mathbf{x})}{g_i(\mathbf{x}) + g_o(\mathbf{x})}\, 1. \qquad (16)$$

Figure 8 gives an illustration. Figure 9, 10 and 11 show examples how this approach can be used to deform different shapes.

To steer the inner deformation, boundary constrained modeling approaches use a handle which can freely be placed. Since our approach is volume-preserving, we do not consider scaling and therefore use a 6-DOF handle defined by a 3D location $\mathbf{h}$, a (normalized) normal vector $\mathbf{m}$, and a (normalized) binormal vector $\mathbf{w}$ with $\mathbf{w} \cdot \mathbf{m} = 0$. In order to move a handle $(\mathbf{h}, \mathbf{m}, \mathbf{w})$ to $(\widehat{\mathbf{h}}, \widehat{\mathbf{m}}, \widehat{\mathbf{w}})$, we construct a cubic curve $\mathbf{c}(t)$ and a twisting function $\alpha(t)$ such that the deformation realizes the moving of the handle. We

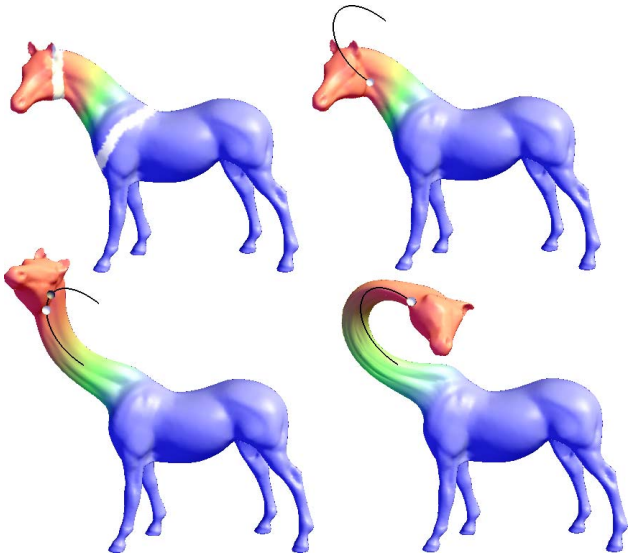

Figure 9. By drawing boundaries (white) onto the neck of the horse, the user can specify the influence regions of the deformation.

Figure 10. Due to the chosen boundaries (white), both the head and the front legs of the cow model undergo a full deformation.

Figure 11. By choosing appropriate boundaries on Armadillo's arm, realistic bending deformations can be obtained.

construct $\mathbf{c}(t)$ as a cubic Bezier curve with the Bezier points

$$\mathbf{b}_0 = \mathbf{h} \quad , \quad \mathbf{b}_1 = \mathbf{h} + \frac{1}{3}(\mathbf{m} \cdot (\widehat{\mathbf{h}} - \mathbf{h})) \cdot \mathbf{m} \qquad (17)$$
$$\mathbf{b}_2 = \widehat{\mathbf{h}} - \frac{1}{3}(\widehat{\mathbf{m}} \cdot (\widehat{\mathbf{h}} - \mathbf{h})) \cdot \widehat{\mathbf{m}} \quad , \quad \mathbf{b}_3 = \widehat{\mathbf{h}}.$$

Figure 12a gives an illustration. Figure 13 shows a rather strong bending deformation of the models.

In order to compute the twisting function $\alpha(t)$, we compute α_0 as the angle between $\mathbf{b}_{2_p} - \mathbf{h}$ and $\mathbf{w}$ where $\mathbf{b}_{2_p}$ is the projection of $\mathbf{b}_2$ onto the plane through $\mathbf{h}$ perpendicular

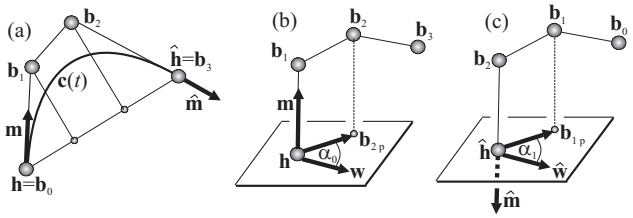

Figure 12. (a) constructing the cubic Bezier curve $\mathbf{c}(t)$ to move $(\mathbf{h}, \mathbf{m}, \mathbf{w})$ to $(\widehat{\mathbf{h}}, \widehat{\mathbf{m}}, \widehat{\mathbf{w}})$; (b)-(c) computing α_0 and α_1 for constructing $\alpha(t)$.

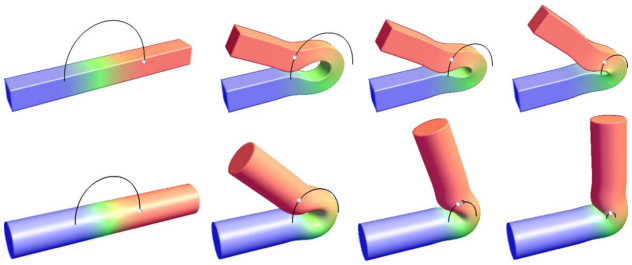

Figure 13. Our approach allows for rather strong bending deformations. The deformation behavior can be controlled by adjusting the radius of the curve.

to $\mathbf{m}$. Figure 12b illustrates this. Furthermore, we compute α_1 as the angle between $\mathbf{b}_{1_p} - \widehat{\mathbf{h}}$ and $\widehat{\mathbf{w}}$ where $\mathbf{b}_{1_p}$ is the projection of $\mathbf{b}_1$ onto the plane through $\widehat{\mathbf{h}}$ perpendicular to $\widehat{\mathbf{m}}$. Figure 12c illustrates this. Then we can compute $\alpha(t) = (1-t)\ \alpha_0 + t\ \alpha_1$ which defines the desired twisting.

In interactive applications, the deformation is not always defined by start and target handle but by interactively moving the handle. Our vector field based approach can deal with this as well: if the handle $(\mathbf{h}, \mathbf{m}, \mathbf{w})$ is simply translated to $(\widehat{\mathbf{h}}, \mathbf{m}, \mathbf{w})$, we can emulate this by constructing $\mathbf{c}(t) = (1-t)\ \mathbf{h} + t\ \widehat{\mathbf{h}}$ and $\alpha(t) =$const. If on the other hand the handle $(\mathbf{h}, \mathbf{m}, \mathbf{w})$ is not moved but only rotated along the axis $\mathbf{h} + \lambda\ \mathbf{t}$ to the handle $(\mathbf{h}, \widehat{\mathbf{m}}, \widehat{\mathbf{w}})$, we can construct a vector field realizing this rotation by choosing $\mathbf{z} = \mathbf{h}$ and (5).

6.3 Knots and extreme deformations

There are deformations which can hardly be achieved by approaches based on a certain energy-minimization. Examples are "knots" in a shape or an extreme twisting. Since our approach steers the inner deformation not by a handle but by a curve and a twisting function, we can handle such deformations. Figure 14 shows an example of making a "knot" into a cylinder model. Figure 15 shows a box which is twisted and at the same time deformed along a curve.

7 Evaluation

In this section we evaluate our approach concerning performance, volume preservation, and possible self-intersections.

7.1 Performance and volume-preservation

In the following table, we see a performance benchmark of our GPU implementation, performed on a GeForce 7800

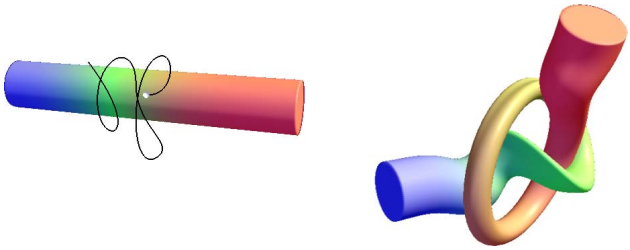

Figure 14. Thanks to the surface-based definition of $r(\mathbf{x}, t)$ and the curve-guided deformation, extreme deformations like knots are possible.

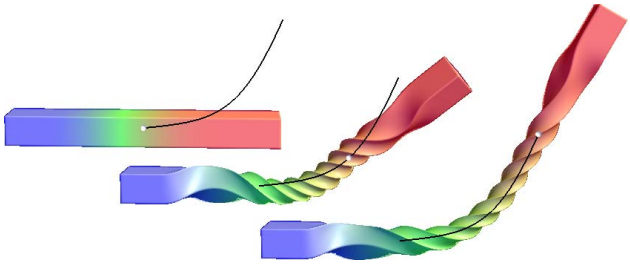

Figure 15. Also extreme twistings combined with a deformation along the control curve are possible.

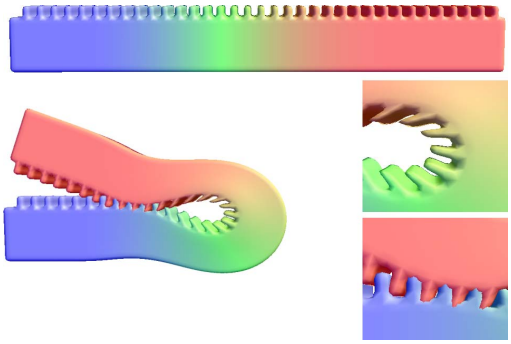

Figure 16. Our deformations tend to prevent local self-intersections, while global self-intersections are possible.

GTX graphics card. In addition, we measured the change of volume with respect to the undeformed shape.

model	fig.	vertices	t/step $[ms]$	vol. error
box	4	2,462	4	1.8%
bar	13	9,730	9	0.4%
squirrel	6	9,995	9	0.08%
horse	9	19,851	22	0.7%
cylinder	14	21,302	20	1.6%
hand	7	53,054	55	0.01%

Since the required number of steps depends on the amount of deformation, we have measured the required time for one integration step. For a complete deformation along a cubic Bezier curve, we used 200 – 300 steps to get accurate results. For the time-per-step measurement, we deformed each model completely, i.e. all vertices were integrated. For the volume measurement, we measured the error after the final deformation depicted in the respective figure, in order to give the reader an illustration.

7.2 Self-intersections

The method in [23] was guaranteed to prevent local and global self-intersections because r, e, f were computed as a global continuous scalar field. Since the method in this paper computes r only as a local approximation of the shape, global self-intersections cannot be excluded any more. For example, our method does not check if parts of the inner region intersect parts of the outer region during the deformation. However, due to the fact that path lines do not intersect in space-time domain, our method can still guarantee that no local self-intersections occur. Figure 16 shows an example of an extreme deformation where global self-intersections occur but local self-intersections are excluded.

While our deformations do not prevent global self-intersections, they are nevertheless volume-preserving with

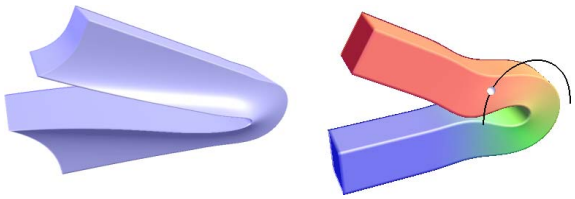

Figure 17. While the original VFSD method (left) prevents global self-intersections, it introduces strong distortions under extreme deformations. Our method (right) tolerates global self-intersections, with the advantage that extreme deformations are free of distortions and the amount of deformation can be exactly controlled.

respect to our definition of volume: Given the shape represented as a closed triangle mesh, we define its volume as the sum of the signed volumes of the tetrahedrons formed by the mesh triangles and the origin. It turns out that our method preserves this volume even during self-intersections. The toleration of global self-intersections has the advantage that more extreme deformations are possible without introducing distortion artifacts. Figure 17 shows a comparison of a strong bending deformation using the original VFSD method and our method. Using the original method, it is hard to control the deformation precisely and strong distortions of the shape occur. Using our method, the volume is preserved more uniformly while the amount of deformation can be precisely controlled.

8 Conclusion

8.1 Contributions

We presented an approach to explicitly control VFSD. In the following, we list the most relevant contributions:

Exact control: In contrast to the approach presented in [23], where the region of influence was controlled by simple implicit objects like spheres or cylinders, our approach permits an exact control of the deformation by defining the deformed regions directly on the surface.

Extended steering of the deformation: Contrary to boundary constraint modeling, the definition of the inner deformation is steered by a curve and a function. This way, deformations can be handled which can hardly be achieved with energy-minimizing approaches.

Efficient deformations on the GPU: In order to make the method suitable for interactive applications, we implemented in on the GPU. The main difference to [23] is that our implementation does not need any read-backs from GPU to CPU during the deformation.

8.2 VFSD features

By utilizing the VFSD technique presented in [23], our approach inherits the following features:

Intuitive editing: Thanks to the VFSD technique, the shape's volume remains constant under deformation. This way, the deformations look natural and help the user to edit shapes in an intuitive manner.

Avoidance of local self-intersections: Due to the nature of path line integration, no local self-intersections can occur during the deformation.

8.3 Limitations

Our deformation technique has some restrictions and limitation which we list below.

Global self-intersections: Contrary to [23], our method cannot guarantee to avoid global self-intersections.

Vertex dependencies: The method in [23] is able to integrate the vertices of the mesh independently of each other. Contrary to this, our approach needs the connectivity information of the mesh to estimate the volumetric field r out of the surface field s.

References

[1] M. Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2):105–114, 2003.

[2] A. Angelidis, M.-P. Cani, G. Wyvill, and S. King. Swirling-sweepers: Constant-volume modeling. In *Computer Graphics and Applications, 12th Pacific Conference on (PG'04)*, pages 10–15, 2004.

[3] A. Angelidis, G. Wyvill, and M.-P. Cani. Sweepers: Swept user-defined tools for modeling by deformation. In *Proceedings of Shape Modeling and Applications*, pages 63–73. IEEE, June 2004.

[4] F. Aubert and D. Bechmann. Volume-preserving space deformation. *Comput. and Graphics*, 21(5):6125–639, 1997.

[5] G. H. Bendels and R. Klein. Mesh forging: editing of 3d-meshes using implicitly defined occluders. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 207–217, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[6] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–491, 2003. (Proceedings Eurographics 2003).

[7] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[8] M. Botsch and L. Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005. (Proceedings Eurographics 2005).

[9] M. Desbrun and M.-P. Gascuel. Animating soft substances with implicit surfaces. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 287–290, New York, NY, USA, 1995. ACM Press.

[10] J. E. Gain and N. A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):289–298, 2001.

[11] G. Hirota, R. Maheshwari, and M. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *Proceedings Solid Modeling and applications*, pages 234–245, 1992.

[12] J. Hua and H. Qin. Scalar-field-guided adaptive shape deformation and animation. *Vis. Comput.*, 20(1):47–66, 2004.

[13] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain mesh deformation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1126–1134, New York, NY, USA, 2006. ACM Press.

[14] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114, New York, NY, USA, 1998. ACM Press.

[15] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel. Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*, pages 181–190. IEEE Computer Society Press, 2004.

[16] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[17] I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. Shaw. Twister: a space-warp operator for the two-handed editing of 3d shapes. *ACM Trans. Graph.*, 22(3):663–668, 2003.

[18] D. Mason and G. Wyvill. Blendeforming: Ray traceable localized foldover-free space deformation. In *CGI '01: Proceedings of the International Conference on Computer Graphics*, page 183, Washington, DC, USA, 2001. IEEE Computer Society.

[19] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.

[20] A. Rappoport, A. Sheffer, and M. Bercovier. Volume-preserving free-form solids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):19–27, 1996.

[21] K. Singh and E. Fiume. Wires: a geometric deformation technique. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 405–414, New York, NY, USA, 1998. ACM Press.

[22] O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 179–188. Eurographics Association, 2004.

[23] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[24] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[25] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel. Harmonic guidance for surface deformation. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, volume 24, pages 601–609, Dublin, Ireland, 2005. Eurographics, Blackwell.

[26] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, 2005.