Shape Deformations
Based on Vector Fields

Wolfram Alexander Freiherr von Funck
Max-Planck-Institut für Informatik

Dissertation zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken, 17. Dezember 2008

Betreuender Hochschullehrer – Supervisor
Prof.Dr. Hans-Peter Seidel,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Gutachter – Reviewers
Prof. Dr. Hans-Peter Seidel,
Max-Planck-Institut für Informatik, Saarbrücken, Germany

Prof. Dr. Holger Theisel,
Otto-von-Guericke-Universität, Magdeburg, Germany

Dekan – Dean
Prof. Dr. Jörg Eschmeier,
Universität des Saarlandes, Saarbrücken, Germany

Datum des Kolloquiums – Date of Defense
22. Dezember 2008

Acknowledgments

I am grateful to Prof. Dr. Hans-Peter Seidel, whose Geometric Modeling lecture was in fact the motivation to carry out my diploma thesis and finally my PhD within his computer graphics group at the MPI Informatik. He gave me the opportunity to work on many interesting projects in a pleasing research environment.

I would like to thank Prof. Dr. Holger Theisel who mentored me during my research. This thesis wouldn't have been possible without his continuing support and motivation.

I also thank Dr. Tino Weinkauf for the good collaboration, as well as the members of the MPI computer graphics group for their advise and ideas.

Finally, I thank my beloved wife Sarah for her interest in my work and for motivating me in hard times.

Abstract

This thesis explores applications of vector field processing to shape deformations. We present a novel method to construct divergence-free vector fields which are used to deform shapes by vector field integration (Chapter 2). The resulting deformation is volume-preserving and no self-intersections occur. We add more controllability to this approach by introducing *implicit boundaries* (Chapter 3), a shape editing method which resembles the well-known boundary constraint modeling metaphor. While the vector fields are originally defined in space, we also present a surface-based version of this approach which allows for more exact boundary selection and deformation control (Chapter 4). We show that vector-field-based shape deformations can be used to animate elastic motions without complex physical simulations (Chapter 5). We also introduce an alternative approach to exactly preserve the volume of skinned triangle meshes (Chapter 6). This is accomplished by constructing a displacement field on the mesh surface which restores the original volume after deformation. Finally, we demonstrate that shape deformation by vector field integration can also be used to visualize smoke-like streak surfaces in dynamic flow fields (Chapter 7).

Kurzfassung

In dieser Dissertation werden verschiedene Anwendungen der Vektorfeldverarbeitung im Bereich Objektdeformation untersucht. Wir präsentieren eine neuartige Methode zur Konstruktion von divergenzfreien Vektorfeldern, welche mittels Integration zum Deformieren von Objekten verwendet werden (Kapitel 2). Die so entstehende Deformation ist volumenerhaltend und keine Selbstüberschneidungen treten auf. Inspiriert von etablierten, auf Randbedingungen beruhenden Methoden, erweitern wir diese Idee hinsichtlich Kontrollierbarkeit mittels *impliziten Abgrenzungen* (Kapitel 3). Während die ursprüngliche Konstruktion im Raum definiert ist, präsentieren wir auch eine oberflächenbasierte Version, welche ein genaueres Festlegen der Abgrenzungen und bessere Kontrolle ermöglicht (Kapitel 4). Wir zeigen, dass vektorfeldbasierte Deformationen auch zur Animation von

elastischen Bewegungen benutzt werden können, ohne dass komplexe Simulationen nötig sind (Kapitel 5). Desweiteren zeigen wir eine alternative Möglichkeit, mit der man das Volumen von Dreiecksnetzen erhalten kann, welche mittels Skelett-Animation deformiert werden (Kapitel 6). Dies erreichen wir durch ein Deformationsfeld auf der Oberfläche, das das ursprüngliche Volumen wieder hergestellt. Wir zeigen ausserdem, dass Deformierungen mittels Vektorfeld-Integration auch zur Visualisierung von Rauch in dynamischen Flüssen genutzt werden können (Kapitel 7).

Contents

1 Introduction

Deforming shapes under a number of constraints is a standard problem in Computer Graphics. For instance, a character animation can be achieved by deforming the shape of the character according to its underlying skeleton. Elastic bodies are deformed by performing simulations based on physical laws. In industrial design, fair surface deformations are obtained by minimizing curvature energy.

In this thesis, we handle the constraint of volume preservation by constructing special vector fields and using these to deform shapes. As it turns out, this approach is both computationally efficient and gives physically plausible results.

In **Chapter 2**, we present a novel method to construct divergence-free vector fields. This mathematical construction method neither requires precomputations nor special data structures like grids or meshes. We show that path line integrations of these vector fields can be a powerful shape deformation tool with several desirable properties: Due to the zero-divergence, the volume of a shape is preserved during deformation. This leads to rather realistic results because the shape appears to consist of a real, incompressible material. The path line integration inherently prevents self-intersections during deformation without the need for complex collision detections. Furthermore, the deformation is defined for all points in space, making it independent of the shape representation. This also makes a GPU implementation straightforward, which is able to deform rather complex meshes in real-time.

Chapter 3 extends this method to be more controllable by introducing implicit boundaries. This allows the user to specify the support regions of the deformation on the shape surface rather than in space. The resulting shape editing system resembles traditional boundary constraint editing methods with the addition of volume preservation and prevention of self-intersections.

Chapter 4 adds even more controllability by performing all computations on the vertices of a triangle mesh and by steering the deformation path by a 3D parametric curve. The advantage of this approach is that the deformation regions can be specified per-vertex. Together with the modifiable deformation path, this enables a more exact control over the resulting deformation. While this surface-based approach abandons the automatic prevention of global self-intersections, it still preserves volume and allows for more extreme deformations without distortions resulting from volume preservation.

In **Chapter 5**, we demonstrate that the idea of vector-field-based shape deformations can also be used to animate secondary motions of elastic incompressible materials. We use simple oscillating mass-spring systems to generate dynamic divergence-free vector fields in and around shapes. By performing a kinematic simulation of the mass-spring systems and by deforming the shape according to the resulting vector field, rather realistic secondary deformations can be added to an existing animation, giving the impression of an elastic, incompressible material. While the method is based on a simple heuristic model, it turns out to be an efficient way to emulate elastic objects in real-time on the GPU.

Chapter 6 introduces an alternative method to preserve the volume of a triangle mesh in a single step by displacing the mesh vertices along a user-defined vector field. The required computation is quite straightforward and involves the solution of a cubic equation, which gives the needed scaling of the displacement field. While the previous numerical integrations only approximately preserve volume, this method is exact since it is a direct closed-form solution. We showed how this idea can be applied to mesh skinning to get realistic skeletal character deformations.

In **Chapter 7**, we use vector field integration to visualize smoke-like surfaces in dynamic flow data sets. Being based on semi-transparent triangle mesh surfaces, the method presented in this thesis is the first one which is able to visualize streak surfaces interactively in time-dependent flow fields. This was possible by avoiding expensive remeshing but by coupling the transparency of the triangles to the distortion of the surface. Thanks to the surface representation, the resulting smoke visualization contains self-shadowing and looks rather sharp and detailed even for coarsely sampled meshes. A similar look wouldn't be possible in real-time for particle-based or volumetric methods. Using the same representation, also time lines, streak lines and wool tufts can be visualized.

2 Vector-Field-Based Shape Deformations

Shape deformations is a well-researched area in computer graphics and animation with many applications ranging from automotive design to movie production. A variety of techniques have been developed to transform an original shape into a new one under a certain number of constraints. These constraints can be for instance performance, detail preservation, feature preservation, volume preservation, avoidance of (local or global) self-intersections, or local support. In addition, different metaphors for an intuitive definition and handling of the deformation exist, like the free movement of certain handles [SF98, BK03a, PKKG03], a two-handed metaphor [LKG$^+$03], or the movement of a 9 dof object [BK04].

Most existing deformation approaches have in common that they are defined as a map from the original to the new shape, i.e. there is no information about intermediate deformation steps. For many applications, the user wants to explore the deformation in an interactive manner, i.e. he wants to see a smooth change from the original to the desired shape moving along certain paths. This means that the deformation has to be recomputed again and again at interactive frame rates. To do so, parts of the deformation can be precomputed and reused for every intermediate deformation, see for instance [BK04, BK05].

We introduce an alternative approach to describe shape deformations. We assume that the shape is given as a triangle mesh. We construct a C^1 continuous divergence-free 3D time-dependent vector field $\mathbf{v}$ and obtain the new positions

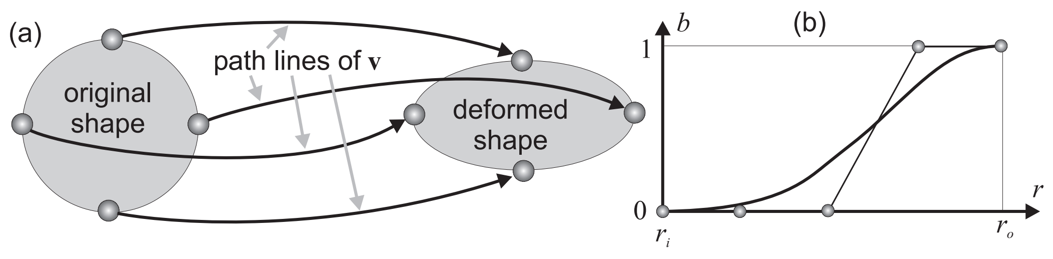


Figure 2.1: (a) Vector-field-based shape deformation: every vertex of the original shape undergoes a path line integration of **v** to find its new position. (b) Blending function $b(r)$.

of every vertex **p** of the shape by applying a path line integration of **v** starting from **p**. This approach is motivated by two observations. First, it corresponds to the metaphor of smooth deformations by observing the paths of the vertices over time. Second, due to the zero-divergence of **v** we get a number of desired properties of the deformation for free. In particular, the following properties hold:

- No self-intersections (neither local nor global) can occur. This is due to the fact that path lines do not intersect in the 4D space-time domain [TWHS05].
- The deformation is volume-preserving. This is a well-known property of divergence-free vector fields [Dav67].
- The deformation preserves the smoothness of the shape to first order. This is due to the C^1 continuity of **v**: Under a path surface integration, the normals of the evolving shape depend on $\nabla\mathbf{v}$. Hence, for a C^1 continuous **v** no discontinuities of the surface normals appear during the integration.
- The deformation preserves details and sharp features in the sense that no smoothing due to an energy minimization occurs.

Figure 2.1a gives an illustration of the main idea.

In addition to the above-mentioned properties of **v**, we construct it to be non-zero only in a certain area to obtain a local support of the deformation. Although divergence-free vector fields have been used to model the flow of fluids [FF01], we are not aware of any approach to use them for the interactive deformation of solid shapes.

This chapter is organized as follows: Section 2.1 gives an overview of related methods. Section 2.2 describes how to construct the locally supported divergence-free vector field **v**. Section 2.3 shows a number of modeling metaphors of our technique. Section 2.4 gives implementation details. Section 2.5 gives an evaluation of our technique and compares it with other approaches. Conclusions are drawn in Section 2.6.

2.1 Related work

Current shape deformation approaches can be classified as surface based methods or space deformation methods. Surface based methods define the deformation only on the shape's surface. A common approach is to specify a number of original and target vertices and compute the remaining vertex positions by a variational approach [WW92, Tau95]. Multiresolution methods are well-established because of their ability to speed up computations and preserve features [ZSS97, GSS99, KCVS98, BK04]. More recently, approaches have been proposed which rely on the solution of the Laplace/Poisson equations [Ale03, LSCO$^+$04, SLCO$^+$04, YZX$^+$04, LSLCO05, ZRKS05]. These approaches end up in the repeated solution of a large sparse linear system. Space deformation techniques modify objects by deforming their embedded space. Prominent representatives of this are free-form deformation methods which can be classified as lattice-based [SP86, Coq90, MJ96], curve-based [Bar84, SF98], or point-based [HML92, HHK92]. Different basis functions to define the space deformation have been applied, like radial basis functions [BK05] or swirls [ACWK04]. [ZHS$^+$05] extends the Laplacian approach from surface based techniques to a volumetric approach. An often addressed issue when dealing with space deformations is the avoidance of self-intersections [AWC04, MW01, GD01]. A number of space deformation techniques are designed to be volume preserving in a global [HML92, DG95, RSB96, AB97, ACWK04] or local [BK03b] way. The constraint of volume preservation promises to give physically more plausible and natural deformations. [ACWK04] presented the first space deformation that is both volume-preserving and foldover-free.

2.2 Constructing the vector field $\mathbf{v}$

The construction of the deformation vector field $\mathbf{v}$ is the core of our approach. It must be flexible enough to describe a variety of different deformations, but simple enough to be computed and updated on-the-fly. We present the construction of $\mathbf{v}$ both for 2D and 3D. While we use the 3D case for our applications, the 2D case serves mainly for illustrating the concept. Also, we formulate the construction in the static (time-independent) context because the extension to time-dependent fields is straightforward.

It is a well-known fact [Dav67] that a 2D divergence-free vector field $\mathbf{v}$ can be constructed as the co-gradient field of a scalar field $p(x,y)$:

$$\mathbf{v}(x,y) = \begin{pmatrix} -p_y(x,y) \\ p_x(x,y) \end{pmatrix}. \tag{2.1}$$

Here, p_x and p_y denote the partial derivatives $\frac{\partial p}{\partial x}$ and $\frac{\partial p}{\partial y}$, respectively. In 3D, a divergence-free vector field $\mathbf{v}$ can be constructed from the gradients of two scalar fields $p(x,y,z)$, $q(x,y,z)$ as

$$\mathbf{v}(x,y,z) = \nabla p(x,y,z) \times \nabla q(x,y,z). \tag{2.2}$$

We are going to construct $\mathbf{v}$ as a piecewise field: inside a certain region, $\mathbf{v}$ should be a simple and well-defined field, such as constant (describing a simple translation of parts of the shape) or linear (describing a rotation). We call this region the *inner region* of the deformation. Also, we have an *outer region* in which we have a zero deformation, i.e., $\mathbf{v} \equiv \mathbf{0}$. Between them there is an *intermediate region* in which $\mathbf{v}$ is blended between inner and outer region in a globally divergence-free and C^1 continuous way. We specify the different regions implicitly by defining a scalar *region field* $r(\mathbf{x})$ with $\mathbf{x} = (x,y)$ or $\mathbf{x} = (x,y,z)$, and two thresholds $r_i < r_o$. Then a point $\mathbf{x}$ is in the inner region if $r(\mathbf{x}) < r_i$, $\mathbf{x}$ is in the intermediate region if $r_i \le r(\mathbf{x}) < r_o$, and $\mathbf{x}$ is in the outer region if $r_o \le r(\mathbf{x})$.

Let $e(\mathbf{x})$, $f(\mathbf{x})$ be two C^2 continuous scalar fields which are supposed to define

$\mathbf{v}$ in the inner region, i.e. $\mathbf{v} = \nabla e \times \nabla f$ there. Then we can define the piecewise scalar fields p, q as

$$p(\mathbf{x}) \;=\; \begin{cases} e(\mathbf{x}) & \text{if} \quad r(\mathbf{x}) < r_i \\ (1-b)\cdot e(\mathbf{x}) \,+\, b\cdot 0 & \text{if} \quad r_i \le r(\mathbf{x}) < r_o \\ 0 & \text{if} \quad r_o \le r(\mathbf{x}) \end{cases} \tag{2.3}$$

$$q(\mathbf{x}) \;=\; \begin{cases} f(\mathbf{x}) & \text{if} \quad r(\mathbf{x}) < r_i \\ (1-b)\cdot f(\mathbf{x}) \,+\, b\cdot 0 & \text{if} \quad r_i \le r(\mathbf{x}) < r_o \\ 0 & \text{if} \quad r_o \le r(\mathbf{x}) \end{cases} \tag{2.4}$$

where $b = b(r(\mathbf{x}))$ is a blending function given in Bézier representation as:

$$b(r) = \sum_{i=0}^{4} w_i \, B_i^4 \left(\frac{r - r_i}{r_o - r_i} \right) \tag{2.5}$$

where B_i^4 are the Bernstein polynomials [Far02], $w_0 = w_1 = w_2 = 0$ and $w_3 = w_4 = 1$. Figure 2.1b illustrates b. Note that in (2.3) and (2.4) the term $b \cdot 0$ can safely be removed. We left it in the equation to show that in the intermediate region, p is a weighted combination of e and 0, and q is a weighted combination of f and 0.

(2.2)–(2.5) give a C^1-continuous divergence-free vector field $\mathbf{v}$ if the scalar fields e, f, r together with the thresholds r_i, r_o are given and e, f, r are C^2-continuous. This is ensured because the blending function b is designed to have a sufficient number of vanishing derivatives at $r(\mathbf{x}) = r_i$ and $r(\mathbf{x}) = r_o$. A proof of this is in Section 2.2.2. Also note that from e, f, r and their first order partials, $\mathbf{v}$ can be computed in a closed form. Section 2.2.2 shows this as well.

We illustrate our concept with a 2D example. Setting the region field $r(x,y) = x^2 + y^2$, $r_i = 1$ and $r_o = 4$, the inner region is the unit circle, while the intermediate region is the ring between the radii 1 and 2. We want $\mathbf{v} = (u,v)^T$ to be constant inside the inner region. To do so, e is the linear field $e(x,y) = v\,x - u\,y$ from which (2.3) and (2.1) give $\mathbf{v}$ in all regions. Figure 2.2a illustrates p in the inner and outer region as height field. Figure 2.2b additionally considers p in the intermediate region. Figure 2.2c shows the resulting $\mathbf{v}$ as a Line Integral Convolution (LIC)

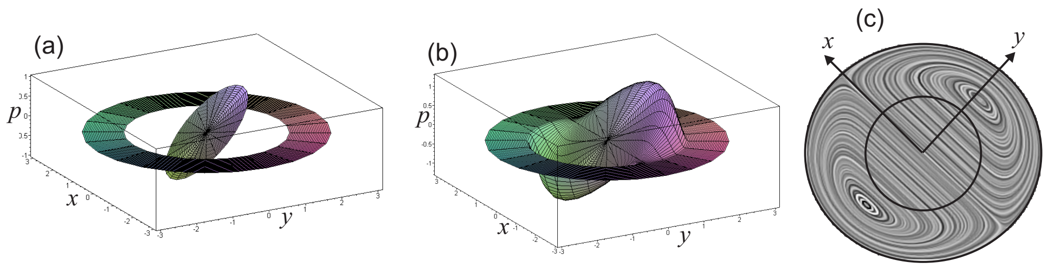


Figure 2.2: Constructing a constant **v** inside the inner region: (a) p as height field in inner and outer region, (b) p in all regions, (c) **v** in inner and intermediate region.

image in the inner and intermediate region. In the outer region, **v** is constant zero.

2.2.1 Special Deformations

Theoretically, every divergence-free vector field **v** can be considered in the inner region, i.e., arbitrary C^2 scalar fields e, f can be chosen. However, for our applications we particularly used constant, linear, and quadratic vector fields.

A constant vector field **v** in the inner region describes a translation inside this region. To get a constant field $\mathbf{v} = (u, v, w)^T$ with $\|\mathbf{v}\| = 1$, we choose two arbitrary orthogonal vectors **u**, **w** with $\|\mathbf{u}\| = \|\mathbf{w}\| = 1$ and $\mathbf{u}\,\mathbf{v} = \mathbf{u}\,\mathbf{w} = \mathbf{v}\,\mathbf{w} = 0$. Also, we have to set a center point **c** inside the inner region. This is necessary because e, f have a constant to be added as degree of freedom. The center point **c** fixes this by setting $e(\mathbf{c}) = f(\mathbf{c}) = 0$. Then the linear scalar fields

$$e(\mathbf{x}) = \mathbf{u}\,(\mathbf{x} - \mathbf{c})^T \quad , \quad f(\mathbf{x}) = \mathbf{w}\,(\mathbf{x} - \mathbf{c})^T \tag{2.6}$$

produce **v** because (2.6) yields $\nabla e \equiv \mathbf{u}$ and $\nabla f \equiv \mathbf{w}$.

A linear vector field **v** is used to describe a rotation inside the inner region. Given a rotational axis by a center point **c** and the normalized axis direction **a**, the field e is linear with the gradient **a**, while the field f is quadratic, describing the squared

Euclidean distance to the rotation axis:

$$e(\mathbf{x}) = \mathbf{a}\,(\mathbf{x}-\mathbf{c})^T \quad , \quad f(\mathbf{x}) = (\mathbf{a}\times(\mathbf{x}-\mathbf{c})^T)^2. \tag{2.7}$$

2.2.2 Continuity

Given the C^2 continuous scalar fields e, f and the C^2 continuous region field r with the thresholds r_i, r_o, we show that $\mathbf{v}$ constructed by (2.2)–(2.5) is divergence-free and C^1. The zero-divergence follows directly from (2.2) [Dav67]. For showing the C^1 continuity, we have to consider the boundaries of the regions, i.e., the locations $\mathbf{x}$ with $r(\mathbf{x}) = r_i$ and $r(\mathbf{x}) = r_o$.

For $r(\mathbf{x}) = r_i$, (2.5) gives

$$b = 0 \quad , \quad \frac{db}{dr} = 0 \quad , \quad \frac{d^2b}{dr^2} = 0. \tag{2.8}$$

For $b = b(r(\mathbf{x}))$, basic rules in differential calculus give

$$\nabla b = \frac{db}{dr}\nabla r \quad , \quad \mathbf{J}(\nabla b) = \frac{db}{dr}\mathbf{J}(\nabla r) + \frac{d^2b}{dr^2}\nabla r \nabla r^T. \tag{2.9}$$

To prove that $\mathbf{v}$ is C^1, we have to show

$$\nabla e \times \nabla f = \nabla((1-b)\,e) \times \nabla((1-b)\,f) \tag{2.10}$$

$$\mathbf{J}(\nabla e \times \nabla f) = \mathbf{J}(\nabla((1-b)\,e) \times \nabla((1-b)\,f)) \tag{2.11}$$

(2.2)–(2.4) give that the left-hand side of (2.10) describes $\mathbf{v}$ in the inner region, while the right-hand side describes $\mathbf{v}$ in the intermediate region. (2.11) does so for the Jacobian of $\mathbf{v}$ in inner and intermediate region. Applying basic rules of differential calculus, we get

$$\nabla((1-b)\,e) = (1-b)\cdot\nabla e - e\nabla b \tag{2.12}$$

$$\nabla((1-b)\,f) = (1-b)\cdot\nabla f - f\nabla b \tag{2.13}$$

and

$$\begin{aligned}
\mathbf{J}(\nabla((1-b)\,e)) &= (1-b)\,\mathbf{J}(\nabla e) \;-\; \nabla e\,\nabla b^T \\
&\quad -e\,\mathbf{J}(\nabla b) \;-\; \nabla b\,\nabla e^T && (2.14) \\
\mathbf{J}(\nabla((1-b)\,f)) &= (1-b)\,\mathbf{J}(\nabla f) \;-\; \nabla f\,\nabla b^T \\
&\quad -f\,\mathbf{J}(\nabla b) \;-\; \nabla b\,\nabla f^T && (2.15)
\end{aligned}$$

Inserting (2.8),(2.9) into (2.12),(2.13),(2.14),(2.15) we get

$$\nabla((1-b)\,e) = \nabla e \quad , \quad \nabla((1-b)\,f) = \nabla f \tag{2.16}$$

$$\mathbf{J}(\nabla((1-b)\,e)) = \mathbf{J}(\nabla e) \quad , \quad \mathbf{J}(\nabla((1-b)\,f)) = \mathbf{J}(\nabla f) \tag{2.17}$$

which gives (2.10),(2.11). In fact, (2.16),(2.17) show that p and q defined in (2.3), (2.4) are C^2 across locations with $r = r_i$.

For $r(\mathbf{x}) = r_o$, (2.5) gives

$$b = 1 \quad , \quad \frac{db}{dr} = 0. \tag{2.18}$$

To prove that $\mathbf{v}$ is C^1, we have to show

$$\nabla((1-b)\,e) \times \nabla((1-b)\,f) = \mathbf{0} \tag{2.19}$$

$$\mathbf{J}(\nabla((1-b)\,e) \times \nabla((1-b)\,f)) = \mathbf{0} \tag{2.20}$$

where the left-hand side of (2.19) describes $\mathbf{v}$ in the intermediate region and the right-hand side in the outer region. Inserting (2.9),(2.18) into (2.12), (2.13) gives (2.19). Inserting (2.9),(2.18) into (2.14), (2.15) together with $\mathbf{J}(\mathbf{a}\times\mathbf{b}) = \mathbf{J}(\mathbf{a})\times\mathbf{b} + \mathbf{a}\times\mathbf{J}(\mathbf{b})$ gives (2.20). It shows that at locations with $r = r_o$, p and q are only C^1, whereas $\mathbf{v}$ is C^1 as well. The C^1 continuity of p and q is sufficient here because we have the additional condition that $\mathbf{v}$ equals zero.

Note that (2.12),(2.13) together with (2.9) and $\frac{db}{dr} = \frac{2r}{r_o - r_i}$ from (2.5) gives the closed form of $\mathbf{v}$ if e, f, r and their first order partials are given.

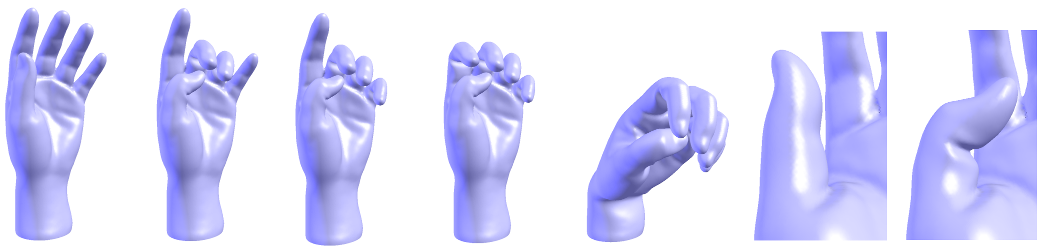


Figure 2.3: Volume-preserving deformation of the hand model (36619 vertices): no skeletal hand model is involved, no self-intersections occur.

2.3 Modeling metaphors

Our approach works as a simultaneous path line integration and updating of $\mathbf{v}$. In fact, at a certain time, for every vertex one integration step of a numerical path line integration of $\mathbf{v}$ is carried out. Then $\mathbf{v}$ is updated, i.e., the defining fields e, f, r together with r_i, r_o are changed before the next integration step is carried out. There are different strategies to define and update $\mathbf{v}$, leading to a number of modeling metaphors of our technique. Before describing them in detail, we explain the visualization of the deformation tools, i.e., $\mathbf{v}$ at a certain time. We represent the region field r by a red semitransparent isosurface $r(\mathbf{x}) = r_i$ and a green surface $r(\mathbf{x}) = r_o$ separating the different regions of the deformation. If we use a constant $\mathbf{v}$ in the inner region (Section 2.2.1), we show it by an arrow whose origin is the central point $\mathbf{c}$ and whose direction denotes $\mathbf{v}$. Figure 2.4a shows an example where r describes the distance to a certain point and $\mathbf{c}$ is in the center of the inner region (red sphere). If we apply a rotation inside the inner region (Section 2.2.1), we show $\mathbf{c}$ and the central axis. If we combine it with a linear r, the isosurfaces $r(\mathbf{x}) = r_i$ and $r(\mathbf{x}) = r_o$ are planes. In order to make the deformation local, we restrict it to a cylinder with the main axis parallel to ∇r and $\mathbf{c}$ on the main axis. Figure 2.4b shows the tool. Since this way the deformation vector field $\mathbf{v}$ is discontinuous across the cylinder barrel, the tool is only applicable if no part of the shape intersects the cylinder barrel at the beginning of the deformation.

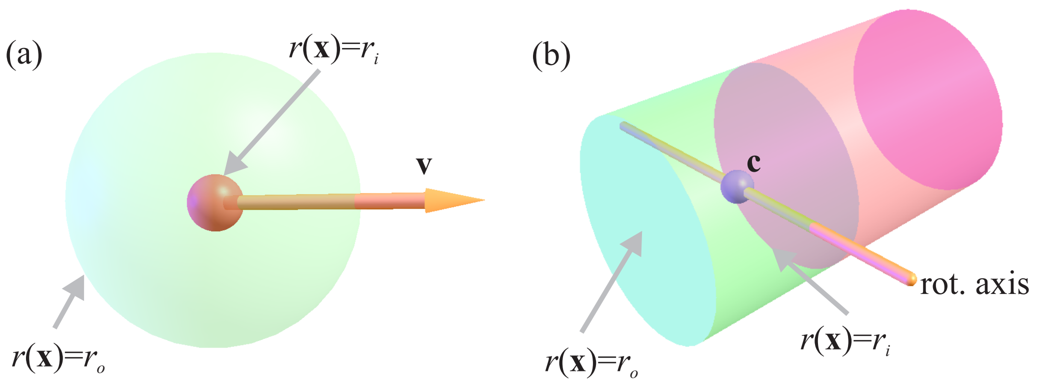


Figure 2.4: Deformation tools: (a) translation, (b) rotation inside the inner region.

2.3.1 Implicit Tools

For the metaphor of implicit tools, we define an arbitrary scalar field r together with r_i and r_o. Usually, r is a simple function describing the distance to a point (Figure 2.5) or a line segment (Figure 2.6). Furthermore, $\mathbf{c}$ is located in the center of the inner region of the deformation. Inside this inner region, $\mathbf{v}$ is constant, describing a translation where its length and direction is determined by position and movement of an interactive input device (e.g. a mouse). When the tool is interactively moved, $\mathbf{v}$ is updated on-the-fly according to the movement. The step size of the path line integration is chosen so that the path line follows the path of the tool: if the interactive motion changes the position of the tool by $\Delta\mathbf{r}$, then the integration inside the inner region moves the points by $\Delta\mathbf{r}$ as well (see Section 2.4.1 for more detail). This way we get the following property: If at the beginning of the deformation parts of the surface are in the inner region of the tool, they follow the path of the tool. Figure 2.5 shows an example. If at the beginning of the deformation the inner region is completely outside the shape, the shape will never enter the inner region. Figure 2.6 illustrates this. Figure 2.7 shows the result of an extreme deformation by moving the tool toward and through the shape: no self-intersection can occur.

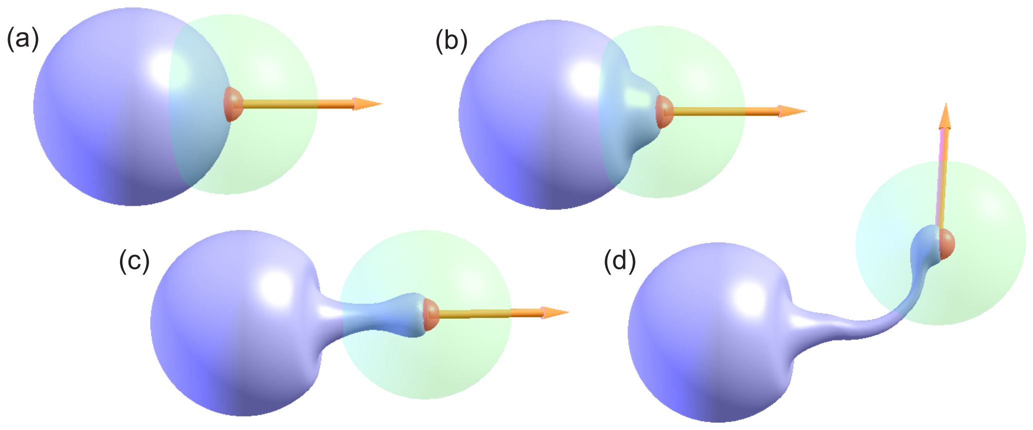


Figure 2.5: Deforming a sphere with an implicit tool: the parts inside the inner region follow the path of the tool.

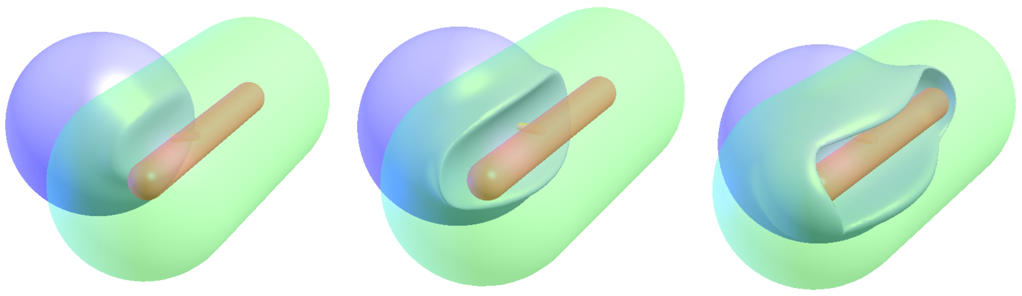


Figure 2.6: Moving the implicit tool toward the shape: the inner region never enters the shape.

2.3.2 Deformation Painting

In this modeling metaphor, the tool is moved along a path on the surface of the shape. For this path, the surface is locally deformed into or out of the shape. If the tool is at the location $\mathbf{x}_s$ on the shape at a certain time, we use $r(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_s\|$, $r_i = 0$, and r_o is interactively chosen to steering the area of influence of the deformation. This means that the inner region is only the point $\mathbf{x}_s$ for which we use a constant $\mathbf{v}$ in the direction opposite to the surface normal of $\mathbf{x}_s$. Figure 6.3 shows an example of deformation painting on a hand model.

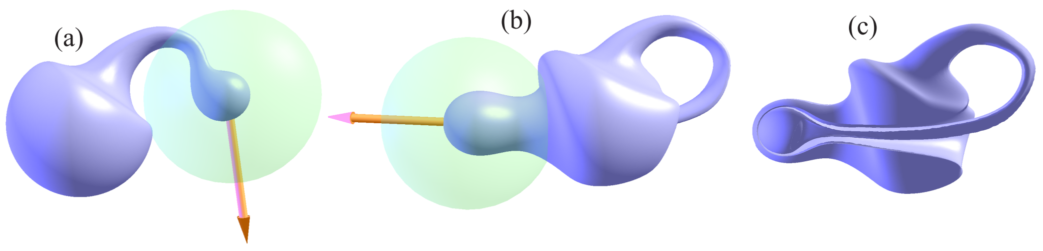


Figure 2.7: (a),(b) Moving an implicit tool through a sphere shape: no self-intersections occur. (c) Same shape as (b) but with cutting plane.

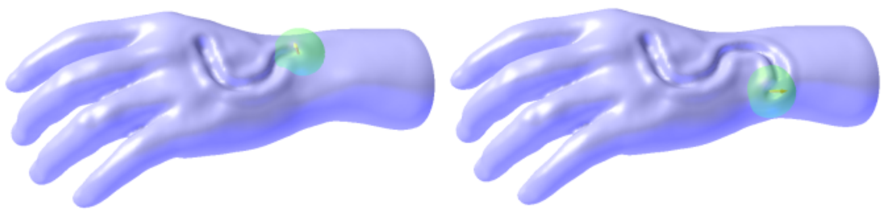


Figure 2.8: Deformation painting on a hand model.

2.3.3 Moving point sets

In this metaphor, we mark a number of points on the shape. These points may be isolated or located on a curve. Then we set r as a smooth approximated distance function to this point set, $r_i = 0$, and r_o is interactively chosen. Inside the inner region, a constant $\mathbf{v}$ is used. For the distance function to the point set, we used the approach described in [BS04]. Figure 2.9 illustrates an example. The barycenter of all points is used as $\mathbf{c}$. Note that in this scenario the inner and the intermediate regions may consist of multiple unconnected parts.

2.3.4 Collision tools and shape stamping

In this scenario, the tool is described by an arbitrary closed tool shape for which a repeated collision detection with the deformed shape is carried out. To do so, we used an approach using bounding box hierarchies implemented in [Col]. Being based on the detected collision points, r is constructed to be zero only in the areas of collision. Similar to moving points sets, we used a smooth approximated

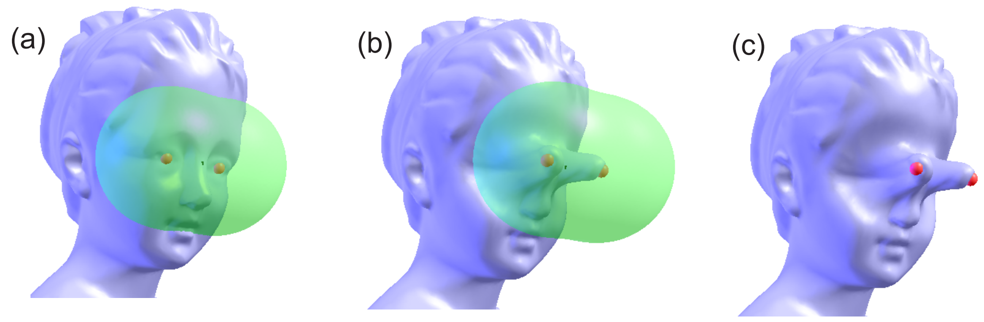


Figure 2.9: Moving point sets: the inner region consist of two unconnected parts close to the eyes of the bust.

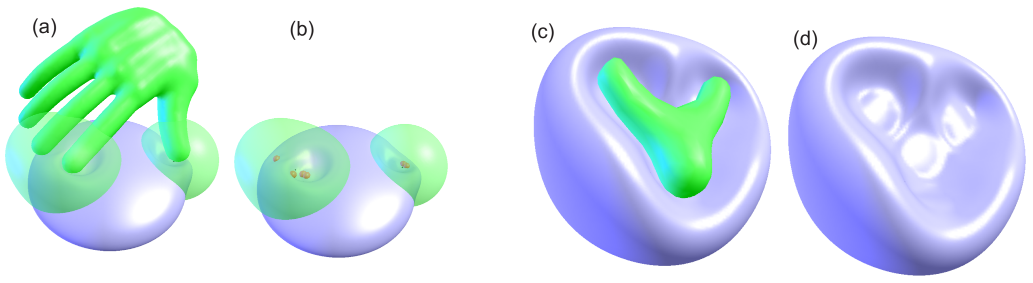


Figure 2.10: Collision tools. (a),(b) The inner and the intermediate region consists of unconnected parts. (c),(d) Shape stamping: a Y-shaped tool is stamped onto a sphere.

distance function for r along with $r_i = 0$. Inside the inner region, $\mathbf{v}$ is constant for every time step, following the path of the input device. Figures 5.6a,b illustrate the region function for the hand shape. Here, both the inner and the intermediate region consist of several unconnected parts. Figure 2.11 shows the deformation of the fan data set using the hand tool. Note that the sharp features are preserved under the deformation.

We also used this modeling metaphor for shape stamping: Moving the tool shape toward the deforming shape leaves the footprint on the deforming shape. Figures 5.6c,d show an example stamping a Y-shaped tool shape onto a sphere. Figure 2.13 shows the deformation of the crater data set by using the Armadillo model as tool.

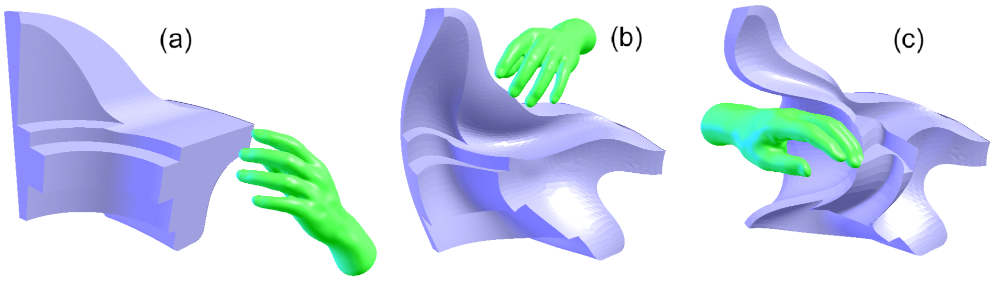


Figure 2.11: Deforming the fan data set.

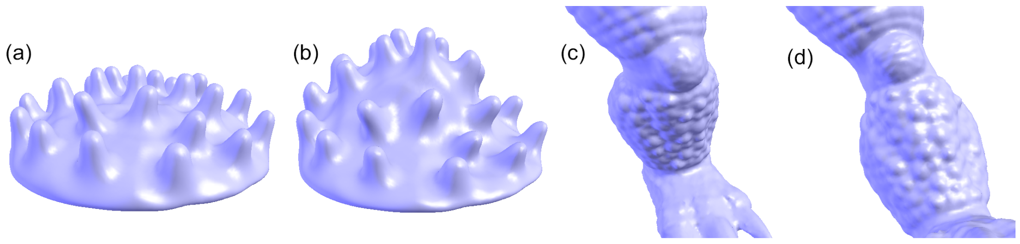


Figure 2.12: Feature preservation: (a),(b) Deforming a model with small details. (c),(d) Bending Armadillo's leg.

2.3.5 Twisting and Bending

Up to now, the vector field inside the inner region was constant. Now we apply linear and quadratic vector fields to get twisting and bending effects. For a twisting, r is linear and its gradient corresponds the direction of the twisting axis. The point $\mathbf{c}$ is on the twisting axis, and the rotational axis for the inner vector field coincides with the twisting axis as well. Inside the inner region we use a linearly increasing rotation defined by $e(\mathbf{x}) = (\mathbf{a}\ (\mathbf{x}-\mathbf{c})^T)^2$, $f(\mathbf{x}) = (\mathbf{a} \times (\mathbf{x}-\mathbf{c})^T)^2$. Figure 2.14 shows an extreme twisting of the box model (51202 vertices) as well as a moderate twisting of the camel model.

To get a bending effect, we used the bending tool described in Figure 2.4b. The region field r is linear and its gradient is perpendicular to the rotation axis. The thresholds r_i, r_o are chosen such that $r(\mathbf{c}) = 0.614r_i + (1-0.614)r_o$. This choice comes from the definition of the blending function: $b(0.614) \approx 1/2$. During the bending, the gradient of r is changed with half the angle speed as the rotation in the inner region. Figure 2.15 gives an example. $e(\mathbf{x})$ and $f(\mathbf{x})$ describe a rotation

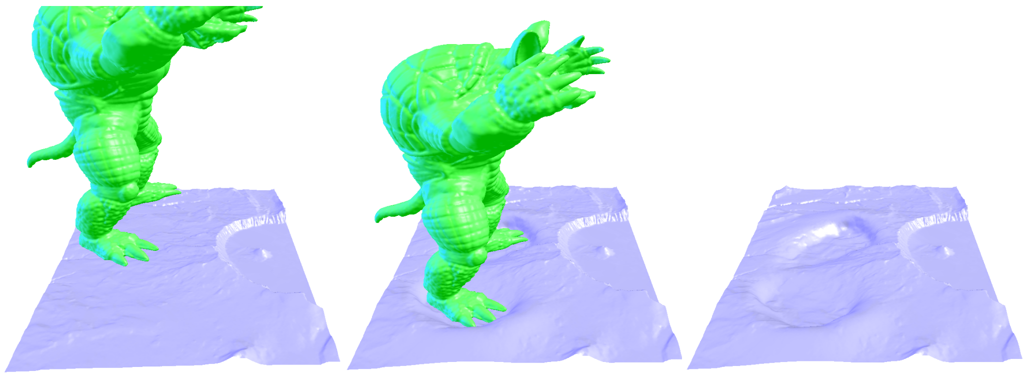


Figure 2.13: Armadillo on the moon: deforming the crater data set with the Armadillo as tool.

(Equation 2.7). Figure 2.3 shows some deformations of the hand data. The result looks rather realistic, even though no skeletal hand model is involved. Figure 2.16 shows the bending of the Armadillo model.

Figure 2.17 shows two shapes created from a sphere in an interactive session by applying all modeling metaphors described above. The session time for each of the models was approximately 30 minutes.

2.4 Implementational Details

2.4.1 Integration with adaptive stepsize

Different approaches for a numerical stream/path line integration have been studied [NHM97], where higher order techniques with an adaptive stepsize turned out to have the best trade-off between speed and accuracy. Thus, in our implementation we used a fourth order Runge-Kutta integration with adaptive stepsize. If a constant stepsize were chosen, the interplay between integration and updating $\mathbf{v}$ is simple: for each vertex, one integration step is carried out, then $\mathbf{v}$ is updated. For an adaptive step size, the synchronization between integration and updating $\mathbf{v}$ is explained in the following example: suppose we use an implicit tool where the mouse moves the central point from $\mathbf{c}_i$ at the time t_i to $\mathbf{c}_{i+1}$. Further assuming that

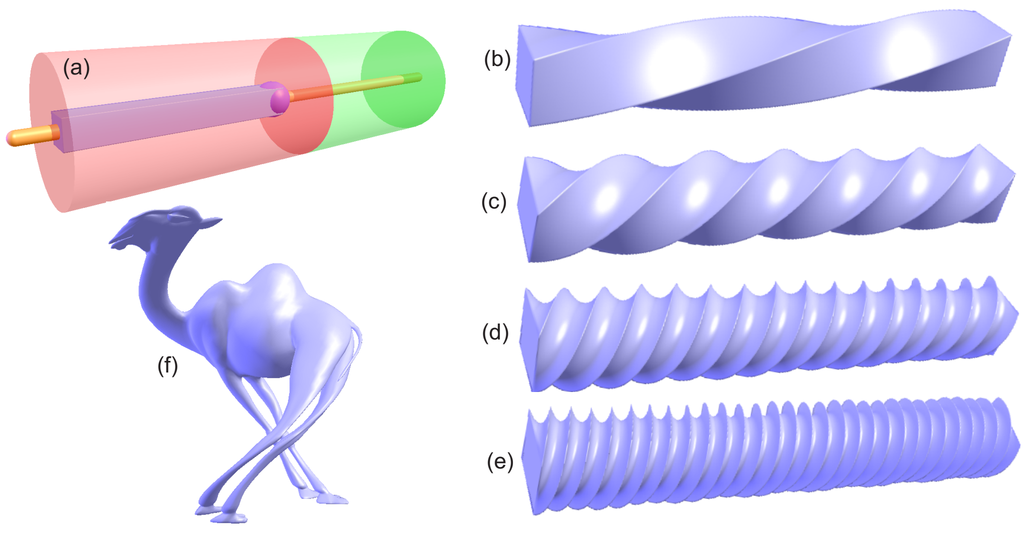


Figure 2.14: Twisting the box model: (a) placing the tool, (b)-(e) twisted models, (f) twisted camel.

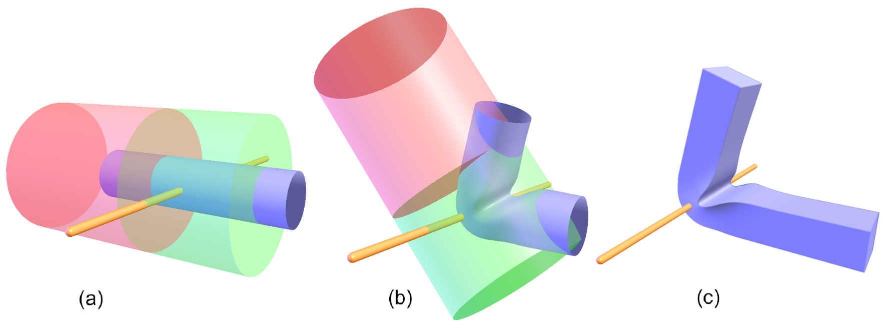


Figure 2.15: (a),(b) Bending a cylinder. (c) Box after bending.

$\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ are in the inner region of the deformation, $\mathbf{v}$ has to fulfill

$$\mathbf{v}(\mathbf{c}_i, t_i) = \mathbf{v}(\mathbf{c}_{i+1}, t_{i+1}) = (t_{i+1} - t_i) \cdot (\mathbf{c}_{i+1} - \mathbf{c}_i) \tag{2.21}$$

Following the description of path lines in [TWHS05], we integrate the 4-dimensional vector field

$$\tilde{\mathbf{v}}(\mathbf{x}, t) = \begin{pmatrix} (1 - \frac{t-t_i}{t_{i+1}-t_i})\mathbf{v}_i + \frac{t-t_i}{t_{i+1}-t_i}\mathbf{v}_{i+1} \\ 1 \end{pmatrix} \tag{2.22}$$

with an adaptive stepsize from a point $(\mathbf{x}, t_i)$ until it reaches a point $(\acute{\mathbf{x}}, t_{i+1})$. In (2.22), t_{i+1} is chosen to fulfill (2.21) and $\mathbf{v}_i$, $\mathbf{v}_{i+1}$, are computed from $r(\mathbf{x}) =$

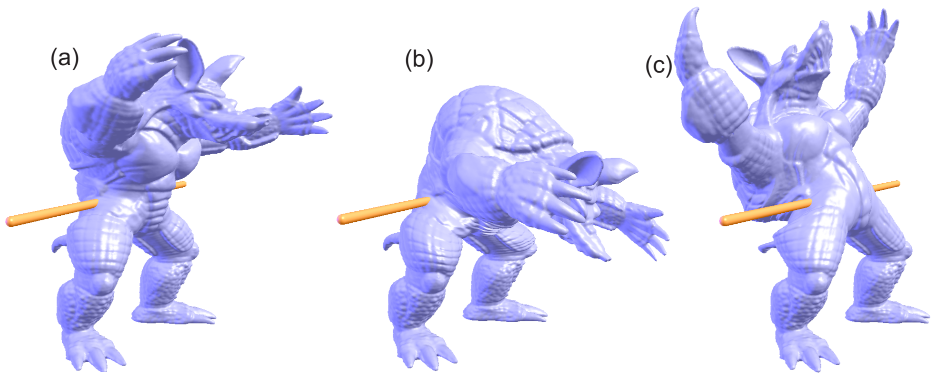


Figure 2.16: Bending the Armadillo model.

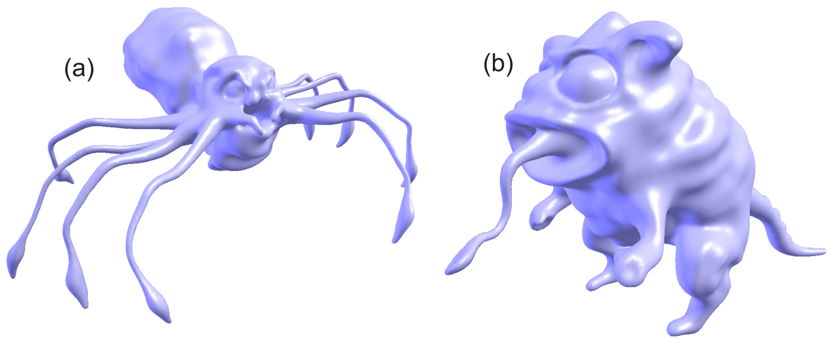


Figure 2.17: Shapes created from spheres in an interactive session.

$\|\mathbf{x} - \mathbf{c}_i\|$ and $r(\mathbf{x}) = \|\mathbf{x} - \mathbf{c}_{i+1}\|$ respectively.

2.4.2 Remeshing

Obviously, large deformations on triangle meshes can cause unpleasing artifacts due to an undersampling of the surface (Figure 2.18a). Furthermore, undersampling can lead to significant volume changes. We deal with this problem by resampling (remeshing) the mesh. For doing so, a variety of approaches exist (see [AUGA05] for a survey). For our algorithm, we used some ideas from [GD99]. The basic idea is to do the remeshing not on the deformed but on the original shape, and let the new vertices undergo the same deformation as the original vertices.

1. While the user performs a deformation on the mesh, the undeformed mesh M and the deformation path (the subsequent translations/rotations) are stored.

2. When the user finishes the operation (in our implementation: when the user releases the mouse button), all edges (ordered by length) of the deformed mesh M' are tested for refinement: if an edge is longer than a certain threshold or the angle between the normals of the end-vertices is large, an edge split is performed both on M and M'. Using the stored deformation path, all new vertices of M are deformed, i.e., path line integrated. Finally, the new vertex positions are copied to the corresponding vertices of M'.

3. In order to guarantee a uniform distribution of the vertices and to eliminate slivers (long, thin triangles), we apply a diffusion of the vertices in M': first, all moved vertices and their immediate neighbors are marked for diffusion. Each marked vertex is moved toward the barycenter of its 1-ring. Afterward, the vertex is projected back onto the surface of the undiffused mesh. This operation is repeated for a fixed number of steps.

4. Subsequent edge splits increase the overall vertex valence and some deformations produce surfaces that are oversampled. Both issues are tackled by a decimation step on M': All edges whose length is smaller than a certain threshold and whose vertex normals enclose a small angle are collapsed at their midpoints.

5. Step 3 is performed again, where the vertices involved in edge collapses are considered as moved vertices.

Figure 2.18 (b) shows a mesh after an application of the above algorithm.

2.4.3 GPU Implementation

Being based on path line integration of single points and being without the need for additional information like mesh connectivity or a skeleton, our deformation approach is highly parallelizable using graphics hardware. We implemented a vertex program to perform an adaptive fourth order Runge-Kutta path line inte-

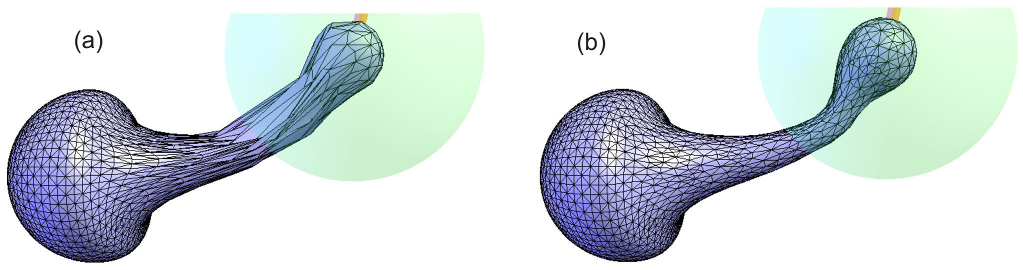


Figure 2.18: Remeshing. (a) Mesh during deformation, (b) mesh after deformation and remeshing.

gration of points. All necessary parameters like translation vector, rotation axis, contact points etc. are passed to the shader as uniform variables. The resulting positions are rendered to a floating point framebuffer. Due to the incremental nature of the algorithm (collision detection after each small deformation, painting on a continually deforming surface etc.) a read-back of the computed points has to be performed after each deformation. Although this drops performance, the computation is still about ten times faster than on the CPU. A detailed performance evaluation can be found in Section 2.5.

2.5 Evaluation and Comparison

In this section we give an evaluation of our technique and compare it with other deformation approaches. We do so in terms of visual quality, other modeling metaphors, speed and accuracy.

Visual quality: To get a comparison with existing techniques, we apply our technique to a number of standard test data sets for which other deformation approaches have been reported in the literature. The twisting of a box (Figure 2.14) has been considered in [YZX$^+$04, LSLCO05, ZHS$^+$05]. Our result shows the behavior of a volume-preserving twisting even for an extreme deformation. The effect of bending a cylinder has been demonstrated for different approaches in [BK03b, BK04, ZHS$^+$05]. Our result (Figures 2.15a-b) shows a realistic looking bend without self intersections. Also, the bending of the box (Figure 2.15c) and the deformation of the hand (Figure 2.3) look plausible and do not contain self-intersections. Furthermore, small scale features are deformed in a plausible manner (Figure 2.12).

Other modeling metaphors: Our implicit tool metaphor using r as the distance to a point (Figures 2.5, 2.7) is similar to the swirling sweepers metaphor [ACWK04]. However, our metaphor is more flexible in the sense that other implicit functions can be used (Figure 2.6). Moreover, contrary to swirling sweepers our method does not have to choose an appropriate number of basic swirls to approximate the final deformation.

Many modeling metaphors [BK04, SLCO$^+$04, BK05] work by setting areas of zero deformation and areas of full deformation on the surface. Then the full deformation is defined by a sequence of translations and rotations. Our tools have a similar metaphor, with the main difference that we define the regions of zero and full deformation implicitly, i.e. by marking the underlying space. This may create problems in areas where surface parts of zero and full deformation are spatially close to each other (for instance the fingers of a hand). For these cases we use the cylinder tool (Figure 2.4b) and restrict the definition to the area inside the cylinder.

Speed: The performance of our approach depends on a number of factors: the number of vertices inside the inner region of the deformation, the number of vertices in the intermediate region, the chosen modeling metaphor, and the chosen region field r. In general, vertices in the intermediate region are more expensive to integrate than vertices in the inner region because $\mathbf{v}$ has a more complicated form there. Also, a simple r (such as the distance to a point in an implicit tool) gives a higher performance. Finally, for the metaphor of shape stamping, the additional collision-detections drops the performance. To get an evaluation of the performance of our approach, we consider a number of benchmark deformations. We placed an implicit tool describing the distance to a point in such a way that no vertex of the shape is in the outer region (i.e., that all vertices have to be integrated), and that most of the vertices are in the (most expensive) intermediate region. Then we applied a rather strong deformation. Figure 2.19 shows the four benchmark deformations by the initial and final shapes as well as used deformation tools. Here, the sphere-shaped intermediate region (green) has been cut out

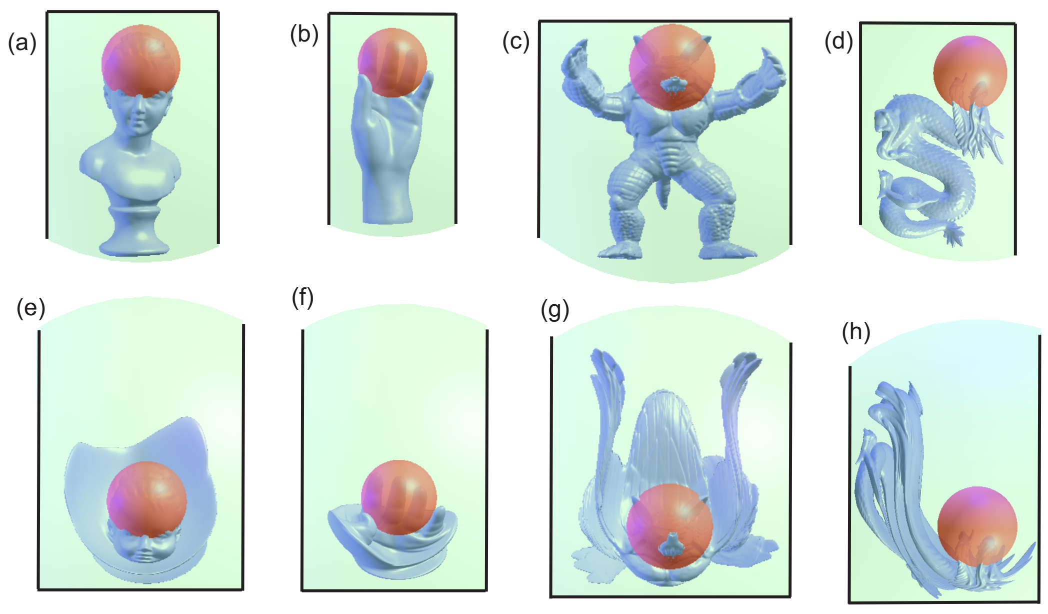


Figure 2.19: Benchmark deformations. First line: original shapes and parts of tools. Second line: deformed shapes and parts of tools.

model	#vert	#steps	sps(CPU)	sps(GPU)
bust	30696	212.737	31.65260	292.2210
hand	36619	186.194	26.94950	257.8860
armadillo	172974	152.656	5.79448	61.8539
dragon	437645	249.196	2.29102	27.2049

Table 2.1: The benchmark shows that even rather large meshes can be deformed interactively.

at the black boundary lines.

Table 2.1 shows the performance on a AMD Opteron 152 (2.6 GHz) with 2 GB RAM and a GeForce 6800 GT GPU. There, #vert denotes the number of vertices of the model, #steps denotes the number of integration steps to come from the original to the final shape, sps(CPU) gives the number of integration steps per second for the CPU implementation, and sps(GPU) does so for the GPU implementation.

It turns out that even for rather large meshes the deformations can be carried out in an interactive manner.

model	orig. shape	deformed shape	*error*
sphere	Fig. 2.5a	Fig. 2.7b	-0.001060
box, twisted	Fig. 2.14a	Fig. 2.14d	0.000781
box, bent	Fig. 2.14a	Fig. 2.15c	0.000751
fan	Fig. 2.11a	Fig. 2.11c	-0.000007
armadillo	Fig. 2.16a	Fig. 2.16c	0.001344
dragon	Fig. 2.19d	Fig. 2.19h	-0.001520
spider	Fig. 2.5a	Fig. 2.17a	0.001875
own monster	Fig. 2.5a	Fig. 2.17b	0.000070

Table 2.2: Measurements show that the volume change is minimal even for strong deformations.

Accuracy: The statement that our approach is volume-preserving holds only if every surface point of the shape undergoes an exact path line integration. In reality, we carry out a numerical integration only for discrete surface points: the mesh vertices. Thus, slight changes of the volume during the deformation can be expected. However, Table 2.2 shows that they are minimal even for strong deformations. Here, we measured the error as $error = \frac{\text{volume(deformed shape)}}{\text{volume(original shape)}} - 1$.

2.6 Discussion

We introduced an alternative approach to shape deformations by carrying out a path line integration of a time-dependent vector field for each shape vertex. This way, simple properties of the vector field lead to useful properties of the deformation: A divergence-free vector field gives a volume-preserving deformation, self-intersections cannot occur, and sharp features are preserved. We have also shown that the performance of the deformation suffices for real-time applications for moderately large meshes. The accuracy of volume preservation is rather high.

There is a number of issues for future research. First, the performance can further be increased by a multi-processor parallelization of the integration. This is possible because the integration of the vertices can be carried out independently of each other. Second, since the method does not rely on any connectivity information of the mesh, an application to point-based shape representations seems possible.

3 Implicit Boundary Control of Vector-Field-Based Shape Deformations

We introduced a method to construct and integrate divergence-free vector fields to obtain volume-preserving deformations without self-intersections. We used rather simple implicitly defined tools like spheres or cylinders to constrain the deformation to certain areas of a shape. In some situations however, a more precise control over the deformation influence is desired. The user often wants to specify precisely which parts of the shape should be deformed and which parts should not be deformed at all. An established method in the area of shape editing is placing boundary constraints on the surface, i.e. the user draws two curves on the surface. The region enclosed by the first curve undergoes a full deformation, e.g. a translation or rotation. The region enclosed by the second curve is not deformed at all. In the region between both curves, the deformation is smoothly blended between full and no deformation.

In this chapter, we present a method which brings both approaches together. The user can define the deformation impact by drawing two boundaries onto the surface of the shape and can deform the shape in a volume-preserving and foldover-free manner with respect to these boundaries. While in most existing approaches boundaries are constraints of an optimization, we introduce implicit boundaries, which are defined by closed polygons and give a direct mathematical solution of a smooth blending function which defines the amount of deformation for every point in space.

This chapter is organized as follows: In Section 3.1, we briefly review the vector field construction method. Section 3.2 describes how implicit boundaries are defined and how they can be used together with vector-field-based shape deformations to deform triangle meshes. Section 3.3 demonstrates several application scenarios, while Section 3.4 goes into the details of the implementation and analyzes the performance. Finally, Section 3.5 discusses the presented method and possible future research.

3.1 Vector Field Construction

In the previous chapter, we introduced vector-field-based shape deformations as a tool with several desirable properties. Due to the C^1 continuity of the vector fields, the resulting deformation is smooth. Due to the path line integration, self-intersections are prevented. Since the vector fields are divergence-free, the volume of the shape remains constant under deformation. Thanks to the direct mathematical formulation of the vector fields, the deformation is independent of the shape representation, requires neither special control structures nor precomputations. Let's briefly review the method in a time-independent context – the extension to the time-dependent case is straightforward.

Given two C^2 continuous scalar fields $p, q : \mathbb{R}^3 \to \mathbb{R}$, a C^1 continuous divergence-free vector field $\mathbf{v}$ can be constructed from the gradients of p and q as

$$\mathbf{v}(x,y,z) = \nabla p(x,y,z) \times \nabla q(x,y,z). \tag{3.1}$$

Simple deformations can be constructed with this method using linear or quadratic scalar fields. In particular, a translation can be achieved by using two linear fields

$$e(\mathbf{x}) = \mathbf{u}(\mathbf{x}-\mathbf{c})^T, f(\mathbf{x}) = \mathbf{w}(\mathbf{x}-\mathbf{c})^T, \tag{3.2}$$

where $\mathbf{u}$ and $\mathbf{w}$ are orthogonal normalized vectors and $\mathbf{c}$ is an arbitrary center point. Since $\mathbf{u}$ and $\mathbf{w}$ are the gradients of e and f, respectively, $\mathbf{u} \times \mathbf{w}$ defines the translation direction. A rotational vector field can be constructed from a linear and a quadratic field:

$$e(\mathbf{x}) = \mathbf{a}(\mathbf{x} - \mathbf{c})^T, f(\mathbf{x}) = (\mathbf{a} \times (\mathbf{x} - \mathbf{c})^T)^2 \tag{3.3}$$

The rotation axis is defined by the normalized vector $\mathbf{a}$, while $\mathbf{c}$ describes the rotation center.

By performing a stream line integration (or path line integration for the time-dependent case) of each mesh vertex in the resulting vector fields, it is possible to rotate and translate a mesh arbitrarily. Obviously, such transformations can be achieved more easily and efficiently by other means. However, we can create more complex and local deformations by blending the scalar functions e, f using a third function b, the *blending function*. In the previous chapter, the blending is done in a piecewise manner, where b is the function of a distance field. Alternatively, we can define b more generally as a C^2 continuous field $b : \mathbb{R}^3 \to [0,1]$. Given b, we can define the blended fields

$$p(\mathbf{x}) = b(\mathbf{x}) \cdot e(\mathbf{x}) \tag{3.4}$$

$$q(\mathbf{x}) = b(\mathbf{x}) \cdot f(\mathbf{x}). \tag{3.5}$$

Using (3.2) for translations or (3.3) for rotations, we can insert (3.4) and (3.5) into (3.1) to obtain a divergence-free vector field, which describes

- a full translation/rotation at points where $b(\mathbf{x}) = 1$,
- a zero-deformation where $b(\mathbf{x}) = 0$,
- a smoothly blended deformation for points where $0 < b(\mathbf{x}) < 1$.

By defining an appropriate blending function b, it is possible to specify which points in space should be deformed by what amount. Figure 3.1 demonstrates this in the 2D setting. Since the blending is done before the cross product of the gradients is computed (Equation 3.1), the resulting vector field is still divergence-free. Since p and q are C^2 continuous, the resulting vector field $\mathbf{v}$ is C^1 continuous

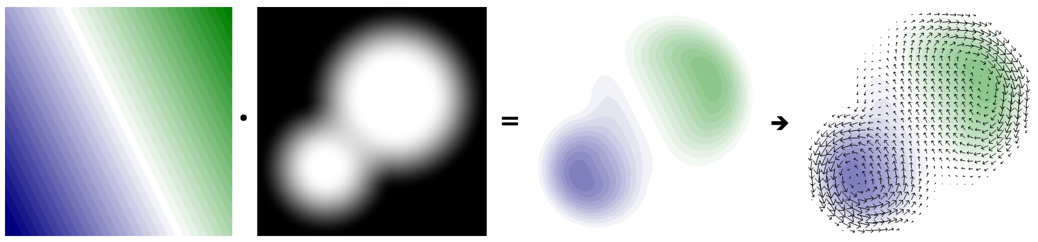


Figure 3.1: Blending the deformation. A linear field (left), describing a translation along its isolines, is multiplied with a blending function (center left). The result is a blended field (center right) from which a divergence-free deformation field (right) can be computed.

(see Chapter 2).

3.2 Deformation Blending

In the following sections, we will show a method to construct a blending function which can be used for boundary constraint modeling using vector-field-based shape deformations.

3.2.1 Implicit Boundaries

From a technical point of view, we don't use the term *boundary constraints* in the sense of an optimization problem, but use boundary constraints as user defined positions in space where the surface should be deformed in a prescribed manner. From the user's point of view, our approach resembles other boundary constraint modeling approaches: the user draws two boundaries on the surface, where the outer boundary defines the support of the deformation, while the inner boundary defines the control handle of the deformation. Figure 3.2 illustrates this.

Since the method of vector-field-based shape deformations is not a surface-based technique but describes space deformations, the boundaries have to be defined in space and not only on the surface. Therefore, we formulate them implicitly. We do so by constructing a smooth implicit function for both the inner and the outer

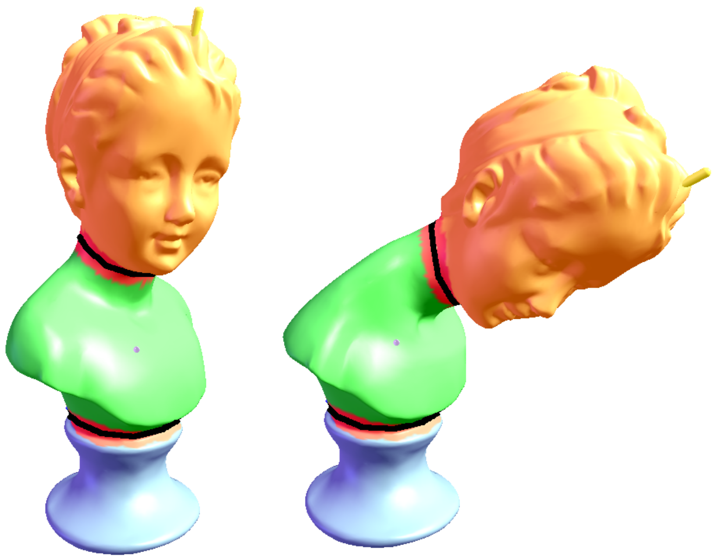


Figure 3.2: By drawing boundaries onto the surface, the user can define a support region (here the body) and a handle region (head).

boundary. More precisely, we define for each boundary a closed piecewise linear curve, i.e. a ring of connected line segments. Then we use an approximate smooth distance field to each curve as implicit function. Given n points $\mathbf{b}_j$, $1 <= j <= n$ with $\mathbf{b}_{n+1} := \mathbf{b}_1$, defining such a polygonal curve, we can compute an approximate distance field using the technique from [BS04] as follows: Given the Euclidian distance fields $l_j(\mathbf{x})$ of each line segment defined by the endpoints $\mathbf{b}_j$, $\mathbf{b}_{j+1}$, we get

$$d(\mathbf{x}) = \frac{1}{\sqrt[k]{\sum_{i=1}^{n} \frac{1}{(l_i(\mathbf{x}))^k}}} \tag{3.6}$$

This corresponds to the *R-equivalence* $l_1(\mathbf{x}) \sim \ldots \sim l_n(\mathbf{x})$ described in [BS04], which joins the distance fields $l_j(\mathbf{x})$ to a smooth approximate one. k is a positive integer, which basically controls the “exactness” of the distance field: the greater k, the more the approximation approaches the real distance field of the polygonal curve, which, in general, contains discontinuities. The smaller we choose k, the smoother the approximation becomes. In our implementation, we use $k = 2$ in order to obtain smooth deformations even for coarse polygons.

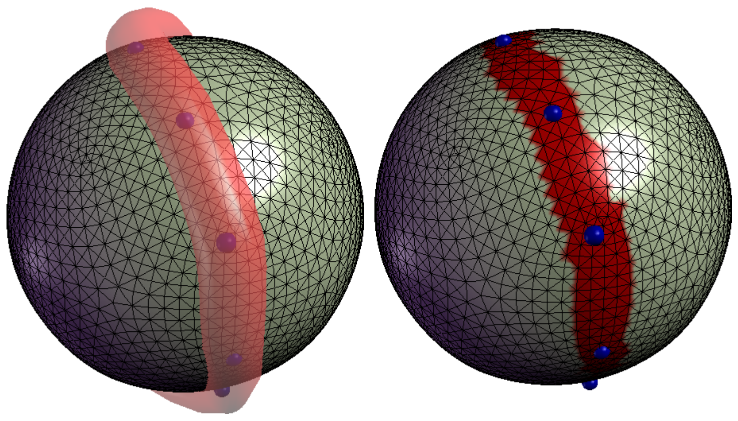


Figure 3.3: Left: an implicit boundary defined by a few points. Right: the corresponding vertices marked.

Given such a scalar field for both the inner and the outer boundary, i.e. $d_i(\mathbf{x})$ and $d_o(\mathbf{x})$, we want to constrain the deformation as follows:

- if $d_i(\mathbf{x}) = t_i$, we want full deformation,
- if $d_o(\mathbf{x}) = t_o$, we want no deformation,
- else, we want a smooth blending between full and zero deformation.

t_i and t_o are user-adjustable thresholds which define the thickness of the innner and outer boundary, respectively. Visually, an implicit boundary can be seen as a closed tube with adjustable thickness running over the surface, as depicted in Figure 3.3. Besides the necessary implicit formulation, this has the advantage that the number of polygon vertices is independent of the mesh resolution, and the user can define smooth boundaries with only a few points, which is especially useful for parallel computation on the GPU. In order to avoid discontinuities in the deformation, the user has to avoid intersections between inner and outer boundary. For instance, when deforming the Dragon's mouth as in Figure 3.4, the boundary thickness has to be chosen such that the boundaries don't touch each other between upper and lower jaw. The points $\mathbf{b}_j$ and t_i, t_o have to be chosen such that the boundary area on the surface is connected, i.e., it divides the shape into two parts. For given t_i, t_o, this can always be achieved by increasing the density of the control points $\mathbf{b}_j$ and by placing them close to sharp features.

3.2.2 Smooth Blending

Having the implicit boundaries defined, we need to construct a function that can be used to blend smoothly from full deformation to zero deformation between inner and outer boundary. This can be accomplished in a straightforward way by interpolation with inverse distance weighting [She68]. We define the blending function as

$$b(\mathbf{x}) = \frac{\frac{1}{(d_i(\mathbf{x})-t_i))^3} \cdot 1 + \frac{1}{(d_o(\mathbf{x})-t_o))^3} \cdot 0}{\frac{1}{(d_i(\mathbf{x})-t_i))^3} + \frac{1}{(d_o(\mathbf{x})-t_o))^3}} \tag{3.7}$$

In the limit, the following holds: $b(\mathbf{x}) = 1$ for $d_i(\mathbf{x}) = t_i$ and $b(\mathbf{x}) = 0$ for $d_o(\mathbf{x}) = t_o$. Because of the cubic weights, $b(\mathbf{x})$ has two vanishing derivatives at points with $d_i(\mathbf{x}) = t_i$ or $d_o(\mathbf{x}) = t_o$. As we will see later, this is an important property which is needed to perform the deformation of the mesh in a piecewise manner. A further degree of freedom can be achieved by multiplying the weights with user-defined factors. This is especially useful to control bend deformations.

So far, we have a C^2 continuous blending function b which can be used together with Equations (3.4), (3.5) to construct blended scalar fields with (3.2) for translations or with (3.3) for rotations. Using (3.1), we can construct a divergence-free vector field, which deforms the mesh (nearly) according to the boundary constraints. Vertices on the inner boundary, i.e. with $d_i(\mathbf{x}) = t_i$, are fully deformed because for them $b(\mathbf{x}) = 1$ holds. Vertices on the outer boundary, i.e. with $d_o(\mathbf{x}) = t_o$, are not deformed because for them $b(\mathbf{x}) = 0$ holds. For all other vertices, the deformation is smoothly blended between full and zero deformation.

3.2.3 Piecewise Deformation

By simply applying such a deformation to the whole mesh, vertices outside of the support region will be deformed as well and vertices in the handle region won't undergo a constant deformation, in general. We therefore need to perform the deformation in a piecewise fashion, which is quite simple. Vertices belonging to the handle region and the inner boundary are deformed in full, i.e. they undergo a constant translation or rotation (e.g. the head and the boundary on the neck in Figure 3.2). Vertices in the support region (body between the boundaries in Figure 3.2), not belonging to any boundary, are deformed using the blended fields (3.4), (3.5). All other vertices are not deformed at all. Thanks to the two vanishing derivatives of the blending function at the boundaries, this piecewise deformation is C^1 continuous. Furthermore, the property of volume preservation still holds as long as no self-intersections occur. Self-intersections can only occur between the deforming parts of the mesh and the non-deforming parts.

Mathematically, we also could instead define the blending function in a piecewise fashion, such that the resulting deformation would be exactly the same. Technically, deforming the mesh in the piecewise manner described above is more efficent because the handle region can be deformed directly using a rigid transformation and the zero-deformation vertices are not considered at all.

3.2.4 Integration in Space Time

The description of the blending function is based on a time-independent context. However, since the inner boundary is actually moving over time, the blending function has to be updated within each integration step. This is straightforward: at the beginning of the integration, the inner boundary polygon is at its original position. Then, with each integration step, the position is updated by the amount corresponding to the step size. For instance, for a rotation, the polygon points are rotated step by step until the full rotation is reached.

3.3 Applications

In principle, every deformation that is constructable as the cross product of two gradients can be used with the described approach. However, we confine ourselves to two simple, yet effective transformations: rotation and translation. Obviously, a scaling transformation wouldn't make sense, since we want to preserve the shape

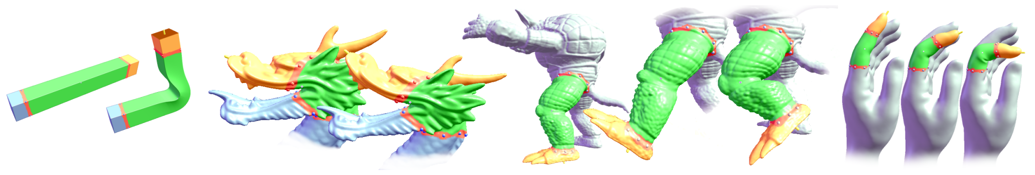


Figure 3.4: Popular application scenarios for boundary constraint modeling.

volume. As we will see in this section, this toolset allows for a variety of useful deformations.

In order to control translation and rotation, the user can place a *knob* somewhere on the mesh surface, and a *joint* somewhere in space. In most of the figures, the knob is depicted as a yellow stick, e.g. on top of the bust in Figure 3.2. The joint is a small white sphere, usually placed somewhere in the support region. The knob resembles typical *Gizmo* objects found in many shape modeling systems, which can be used to control transformations by grabbing and dragging it at different points. The knob is a simplified version because only translation and rotation are supported.

3.3.1 Rotation

In order to rotate the handle region, the user drags the knob, where the knob movement is constrained to a fixed radius about the joint position. From the displacement of the knob position, the rotation axis and angle can be determined with respect to the joint position. Using the joint position as rotation center $\mathbf{c}$ in (3.3), a deformation that bends the shape can be accomplished by integrating the mesh vertices until the rotation angle is reached. In contrast to Chapter 2, where the shape is continually updated, the integration restarts from the original mesh every time the knob position changes. Figure 3.4, as well as Figure 3.2, shows this approach applied in various scenarios known from the Literature. The deformation looks rather realistic thanks to the volume-preservation and avoidance of self-intersections and even high resolution models can be deformed interactively.

Figure 3.5 demonstrates how local details are deformed: The “teeth” of the comb-

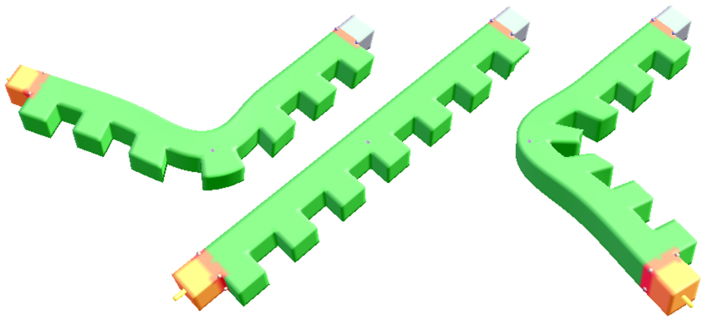


Figure 3.5: Local details are slightly distorted in strongly deformed areas (left) and never intersect with each other (right).

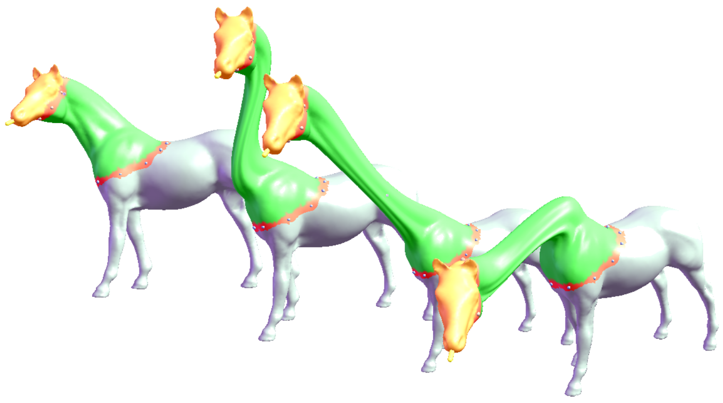


Figure 3.6: By translating the horse head, the neck deforms in a natural manner.

like shape don't touch each other during deformation and their distortion seems appropriate with respect to the global deformation.

Also twisting is possible by simply using the vector between joint and knob as rotation axis. A more uniform twisting deformation can be achieved with two quadratic scalar fields as described in Chapter 2.

3.3.2 Translation

When the user wants to translate the handle region, he or she can drag the knob freely in space. The joint is ignored for this deformation type. However, also the translation (3.2) requires a center point **c**. In this case we use the barycenter of the

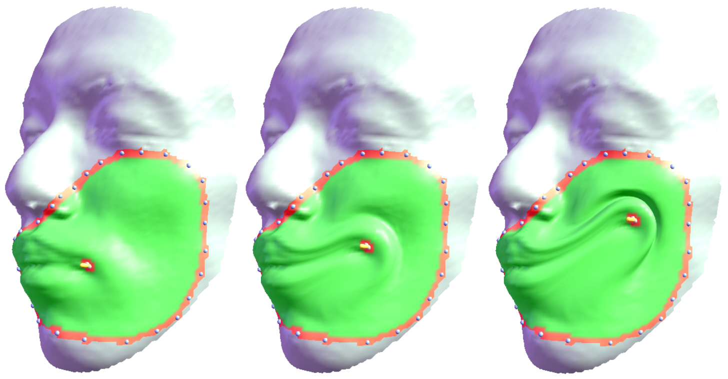


Figure 3.7: “Wrinkles” can be produced by translating the handle accross the surface. Although they appear to be rather strong, no self-intersections of the surface occur.

control points of the inner boundary.

In Figure 3.6, the user drags the head of a horse model. The shape of the neck automatically adapts to the new position. Due to the constant volume, the neck becomes thinner when the head is pulled.

Interesting effects can be achieved by carefully selecting boundaries and moving the handle parallel to the surface: as shown in Figure 3.7, the deformation automatically produces “wrinkles” on the cheek of the face, which is a result of the volume preservation and the prevention of foldovers. Although the “wrinkles” appear to be rather strong, no self-intersections of the surface occur.

3.4 Implementation and Performance

As shown in the previous chapter, the performance of the integration can be increased by a large amount by shifting the computation to the GPU, where multiple path lines can be integrated in parallel. This is also possible with this approach: since the number of boundary control points is usually low, these points can be passed to the shader as a simple array. We implemented two vertex programs, one for translation and one for rotation, which can be controlled by passing translation vector, rotation axis, angle etc. to them. During the integration, the polygon

shape	vertices	boundary points	*v/s* (integration)	*v/s* (integ. + normals)
box	47,296	8	788,267	647,890
dragon	86,814	23	413,400	369,421
leg 1	31,014	17	449,478	382,889
leg 2	31,014	17	443,057	364,871
finger	2725	14	454,167	454,167

Table 3.1: Performance benchmark: complex meshes can be deformed interactively.

points of the inner boundary are updated internally with respect to the translation/rotation, as described in Section 3.2.4. Except for the extraction of the handle and support region (Section 3.2.3), no further preprocessing is required. After integration, vertex positions are read back from video memory and the mesh normals are computed. An alternative approach would be to compute normals directly on the GPU using the Jacobian of the vector field, similar to [BK05]. This would decrease the integration performance because of the necassary computation of Jacobians in each integration step, but would clear the CPU from doing this task and redundatize the readback of vertex positions. However, we have not tested this alternative yet. As we will see in the following performance analysis, the normal computation on the CPU makes only a small fraction of the total deformation time.

The performance of the approach strongly depends on the “amount” of deformation, i.e. how far the handle is translated or rotated. This is because the numerical path line integration adapts its step size according to the complexity of the vector field and the duration of the integration. In order to present a meaningful statement about performance, we measured the times of usual “real-world” deformations, namely the ones depicted in Figure 3.4. Table 3.1 lists the deformed shapes (from left to right in Figure 3.4) and the benchmark results. *v/s (integration)* is the number of vertices per second for integration only and *v/s (integ. + normals)* the number of vertices per second for integration plus normal computation. The measurements were made on a 2.6 GHz Opteron CPU and a GeForce 6800 GT graphics card. They show that complex meshes can be deformed interactively.

3.5 Discussion

We presented a shape deformation technique based on vector field integration which incorporates implicit boundaries to steer the impact of the deformation.

By using vector-field-based shape deformations, our deformations are volume-preserving and foldover-free, giving the user the impression of working with real, incompressible material. While the original approach defined the regions of influence by simple implicit objects, the new method constructs a smooth blending function based on implicit boundaries. That way, the user can specify the impact of the deformation directly on the surface of the shape.

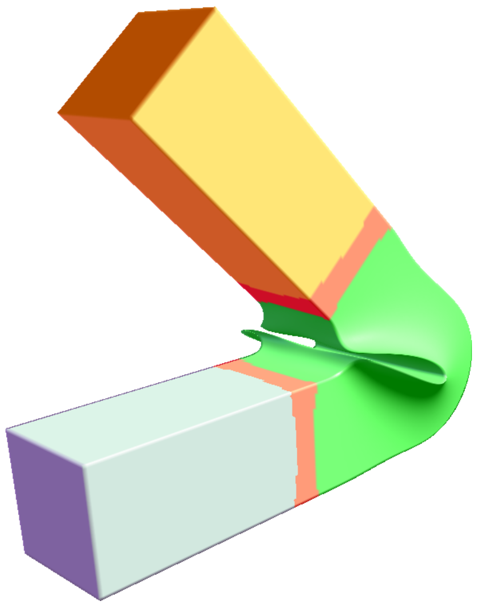


Figure 3.8: When the boundaries move close to each other, the shape is distorted.

Thanks to the polygonal representation of the implicit boundaries, they are independent of the resolution of the deformed mesh. In most cases, a small number of control points suffices to define the boundaries.

Since the description of boundaries is simple (a small set of points), the numerical path line integration can be computed on the GPU and even complex models can be deformed interactively.

The approach has the following restrictions:

Self-intersections. Self-intersections are only avoided for the deforming regions of the shape surface because the deformation is carried out in a piecewise fashion. It is e.g. possible to bend the finger in Figure 3.4 such that the finger tip intersects the thumb or other parts of the hand. An additional collision detection would solve the problem, but would also drop performance.

Close boundaries. When inner and outer boundary are close to each other, the gradient of the resulting blending function is high in these regions. This can result in unpleasing deformations. E.g., when an extreme bending is performed (Figure 3.8), the boundaries approach each other, and the box is distorted more and more at its center (but nevertheless preserves its volume). A possible solution would be to perform such deformations (even more) piecewise, by using for example a third boundary between inner and outer boundary and constructing two blending functions: the first one blends between inner and central boundary, the other one between central and outer boundary.

4 Explicit Control of Vector-Field-Based Shape Deformations

In the previous chapter, we introduced implicit boundaries as a means of more precise control over the deformation. While volume preservation and prevention of self-intersections are obvious advantages over related methods, the method has two drawbacks that we would like to overcome in this chapter.

On the one hand, the specification of influence regions is not per-vertex, i.e. it is not as precise as in comparable boundary constraint shape editing systems. Instead, the user has to specify tube-like boundaries on the surface and has to make sure that all boundary vertices are inside the tube. In contrast to this, existing boundary constraint methods allow for an exact selection of boundary vertices.

On the other hand, strong distortions can occur if two boundaries are located close to each other, as discussed in the previous chapter. This is because of the fact that the blending function and its gradient is defined in space and not only on the shape surface.

In this chapter, we present an approach which defines all fields on the surface of the shape. This has the advantage that a more precise defintion of the deformation boundaries is possible and more extreme deformations can be performed without distortions. In addition, the user can not only define start and end state of the handle region, but also the intermediate steps of the deformation by specifying a

3D parametric curve along which the deformation should be carried out.

Section 4.1 describes our approach from the user's point of view. Section 4.2 shows the underlying vector field construction. Section 4.3 gives details about our GPU-based implementation. Section 4.4 describes different applications, among them a scenario which - from the users's point of view - is similar to boundary constraint modeling. Section 4.5 evaluates our method, while conclusions are drawn in Section 4.6.

4.1 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 the previous method 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 triangle 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 ($s(\mathbf{x}) \leq s_i$) undergoes a full deformation, a vertex $\mathbf{x}$ in the outer region ($s_o \leq s(\mathbf{x})$) remains undeformed, whereas the deformation in the intermediate region ($s_i < s(\mathbf{x}) < s_o$) is obtained by a blending approach. Without loss of generality, we use $s_i = 0$ and $s_o = 1$ throughout this chapter.

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 4.1 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

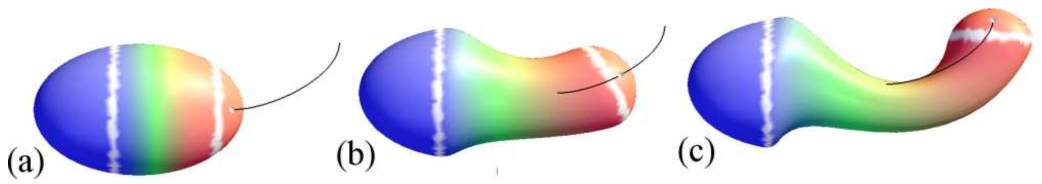


Figure 4.1: (a) The deformation is defined by two closed polygons on the shape and a parametric curve $\mathbf{c}(t)$; (b) path line integration at $t = 1/2$; (c) path line integration at $t = 1$ is the desired deformation.

different regions are highlighted. Also in this and the following images 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.

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 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 4.1 explains an example.

In addition, a twisting effect during the deformation can be obtained 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.

4.2 Constructing the vector field

In this section we describe how to use the method from Chapter 2 with a control of the deformation as described in Section 4.1. It turns out that we can rely on the vector field construction described in (2.2)–(2.5). 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.

4.2.1 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 (2.2)–(2.5). 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 $\mathbf{x}$ and all adjacent vertices in the 1-ring of $\mathbf{x}$. To do so, we apply a least-squares fitting approach of the linear approximation of r in the neighborhood of $\mathbf{x}$: We solve

$$\left[\begin{array}{c} r(\mathbf{x},t) = s(\mathbf{x}) \quad , \quad \nabla r(\mathbf{x},t) \cdot \mathbf{n}(\mathbf{x},t) = 0 \\ \sum_{\mathbf{y} \in R_1(\mathbf{x})} (r(\mathbf{y},t) - s(\mathbf{y}))^2 \rightarrow \min \end{array} \right] \tag{4.1}$$

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, (4.1) 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.

4.2.2 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

$$\mathbf{v}(\mathbf{c}(t), t) = \dot{\mathbf{c}}(t). \tag{4.2}$$

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

$$\nabla e(\mathbf{c}(t), t) \times \nabla f(\mathbf{c}(t), t) = \dot{\mathbf{c}}(t). \tag{4.3}$$

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

$$\begin{aligned} \nabla e(\mathbf{x}, t) \cdot \dot{\mathbf{c}}(t) = \nabla f(\mathbf{x}, t) \cdot \dot{\mathbf{c}}(t) = \\ \nabla e(\mathbf{x}, t) \cdot \nabla f(\mathbf{x}, t) = 0, \quad (4.4) \\ \|\nabla e(\mathbf{x}, t)\| = \|\nabla f(\mathbf{x}, t)\| = \sqrt{\|\dot{\mathbf{c}}(t)\|}, \quad (4.5) \\ e(\mathbf{c}(t), t) = f(\mathbf{c}(t), t) = 0. \end{aligned}$$

Note that (4.5) 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 $\dot{\mathbf{c}}(t)$. However, it turns out that this degree of freedom does not have any influence on the definition of $\mathbf{v}$. Figure 4.2 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

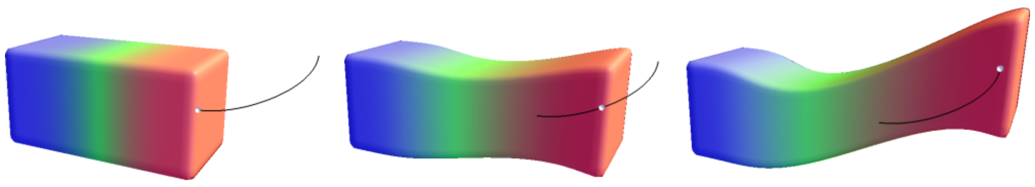


Figure 4.2: Local translation along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$: the inner region follows the curve but does not change its orientation.

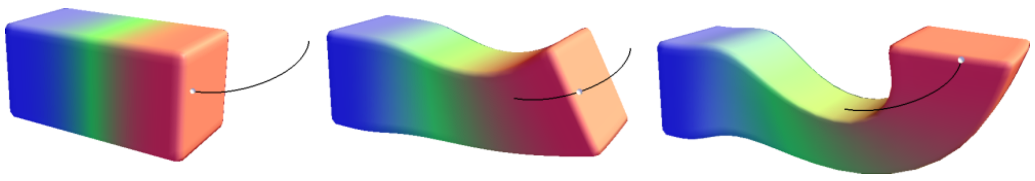


Figure 4.3: Local rotation along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$: the inner region follows the curve both in location and orientation.

compute the curvature center as

$$\mathbf{z}(t) = \mathbf{c} + \frac{\ddot{\mathbf{c}}(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})^2 - \dot{\mathbf{c}}(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})(\dot{\mathbf{c}}\cdot\ddot{\mathbf{c}})}{(\dot{\mathbf{c}}\cdot\dot{\mathbf{c}})(\ddot{\mathbf{c}}\cdot\ddot{\mathbf{c}}) - (\dot{\mathbf{c}}\cdot\ddot{\mathbf{c}})^2} \tag{4.6}$$

and the direction of the rotation axis as

$$\mathbf{t}(t) = (\dot{\mathbf{c}} \times \ddot{\mathbf{c}}). \tag{4.7}$$

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

$$e = \mathbf{t}\cdot(\mathbf{x}-\mathbf{z}) \quad , \quad f = (\mathbf{t}\times(\mathbf{x}-\mathbf{z}))^2 - (\mathbf{c}-\mathbf{z})^2. \tag{4.8}$$

It is straightforward exercise in algebra to show that (4.6), (4.7) and (4.8) give (4.3). Figure 4.3 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 (2.2)–(2.5) by setting $\tilde{r} = r$ and $\tilde{e}, \tilde{f}$ defining a rotation around the axis $\mathbf{c}(t) +$

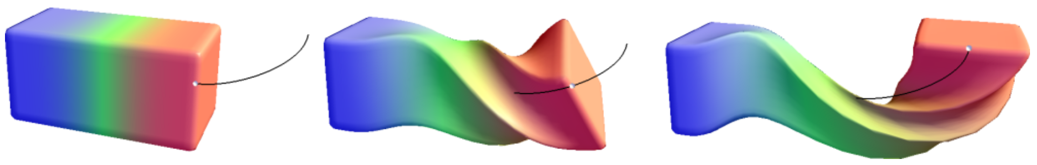


Figure 4.4: Local rotation and twisting along the curve $\mathbf{c}(t)$ for $t = 0, 1/2, 1$.

$\lambda\,\dot{\mathbf{c}}(t)$. In fact, we set

$$\tilde{e}(\mathbf{x},t) = \frac{\dot{\alpha}(t)}{2\pi}(\mathbf{x}-\mathbf{c}(t))\cdot\dot{\mathbf{c}}(t), \tilde{f}(\mathbf{x},t) = ((\mathbf{x}-\mathbf{c}(t))\times\dot{\mathbf{c}}(t))^2. \tag{4.9}$$

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 4.4 illustrates a twisting effect.

4.3 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 necessary 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 4.2.1. 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 4.2 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.

4.4 Applications

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

4.4.1 Geodesic level deformation

The main advantage of the described approach is the fact that we can define the influence of the deformation directly on the surface, while we previously used implicitly defined 3D scalar fields like the Euclidean distance for this purpose. Using a smooth approximation of the geodesic distance to a point on the surface,

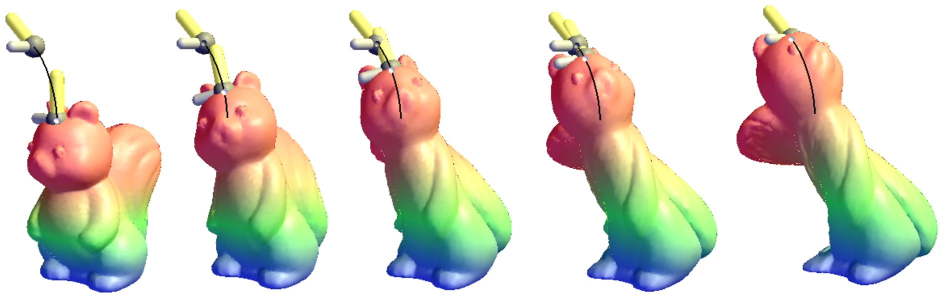


Figure 4.5: The squirrel model is deformed from start to target handle.

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

$$s(\mathbf{x}) = \begin{cases} 1 - \frac{g(\mathbf{x}) - g_{min}}{g_{max} - g_{min}} & \text{if} \quad g_{min} < g(\mathbf{x}) < g_{max} \\ 1 & \text{if} \quad g(\mathbf{x}) \leq g_{min} \\ 0 & \text{if} \quad g_{max} \leq g(\mathbf{x}) \end{cases} \tag{4.10}$$

In Figure 4.6 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.

4.4.2 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 constraint 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 automati-

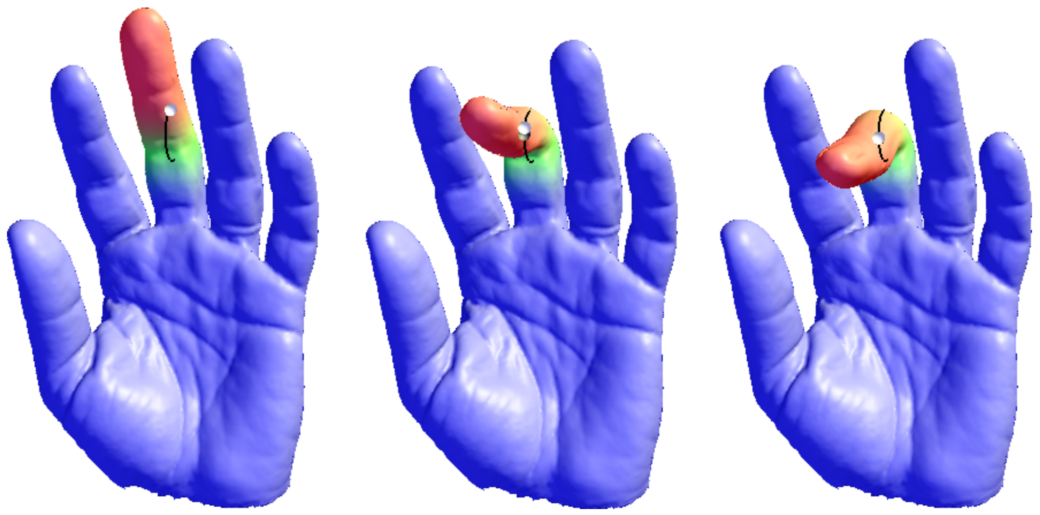


Figure 4.6: Using the geodesic distance field of a point on the tip of the middle finger, we can deform the finger without affecting other parts of the shape.

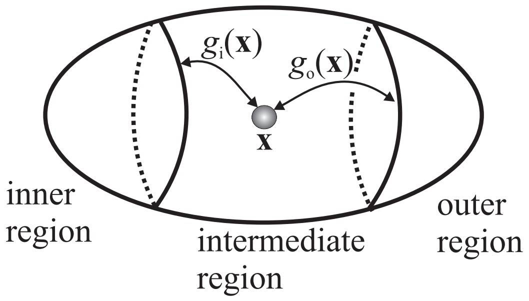


Figure 4.7: Computing $s(\mathbf{x})$ by approximating the geodesic distance between a vertex $\mathbf{x}$ and the inner/outer region.

cally. 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 4.4.1. In 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

$$s(\mathbf{x}) = \frac{g_o(\mathbf{x})}{g_i(\mathbf{x}) + g_o(\mathbf{x})}\,0 + \frac{g_i(\mathbf{x})}{g_i(\mathbf{x}) + g_o(\mathbf{x})}\,1. \tag{4.11}$$

Figure 4.7 gives an illustration. Figure 4.8, 4.9 and 4.10 show examples how this

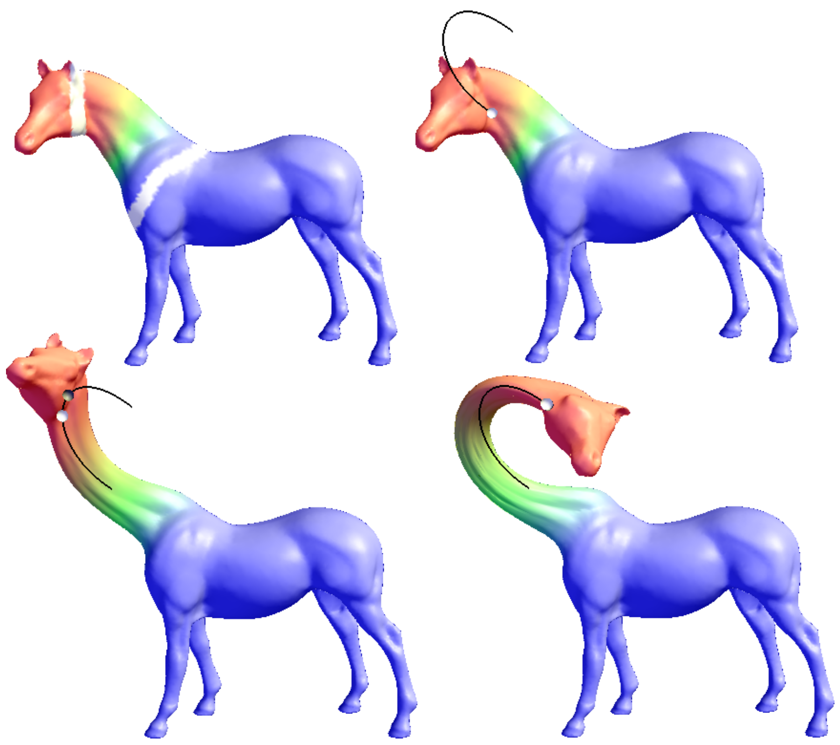


Figure 4.8: By drawing boundaries (white) onto the neck of the horse, the user can specify the influence regions of the deformation.

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 **h**, a (normalized) normal vector **m**, and a (normalized) binormal vector **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 construct $\mathbf{c}(t)$ as a cubic Bezier curve with the Bezier points

$$\mathbf{b}_0 = \mathbf{h} \quad , \quad \mathbf{b}_1 = \mathbf{h} + \frac{1}{3}(\mathbf{m} \cdot (\widehat{\mathbf{h}} - \mathbf{h})) \cdot \mathbf{m} \qquad (4.12)$$
$$\mathbf{b}_2 = \widehat{\mathbf{h}} - \frac{1}{3}(\widehat{\mathbf{m}} \cdot (\widehat{\mathbf{h}} - \mathbf{h})) \cdot \widehat{\mathbf{m}} \quad , \quad \mathbf{b}_3 = \widehat{\mathbf{h}}.$$

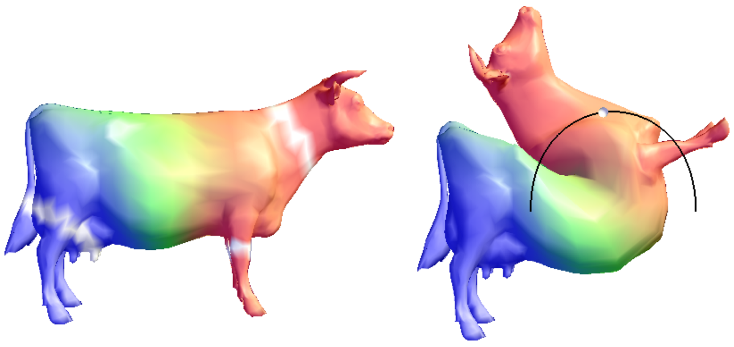


Figure 4.9: Due to the chosen boundaries (white), both the head and the front legs of the cow model undergo a full deformation.

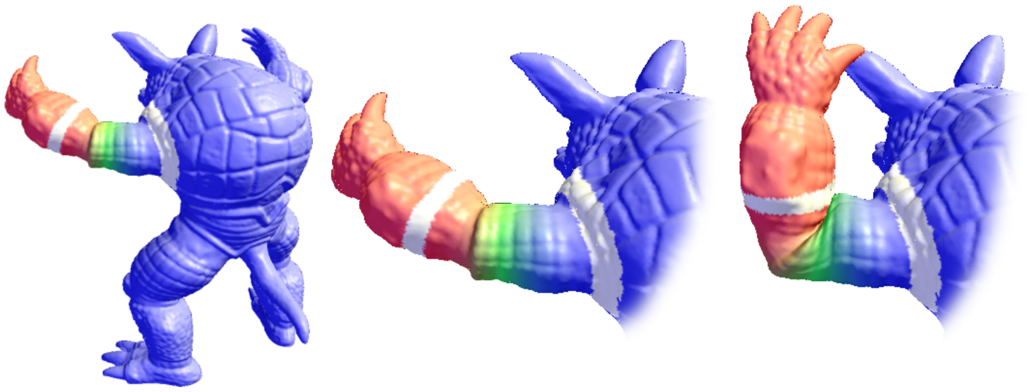


Figure 4.10: By choosing appropriate boundaries on Armadillo's arm, realistic bending deformations can be obtained.

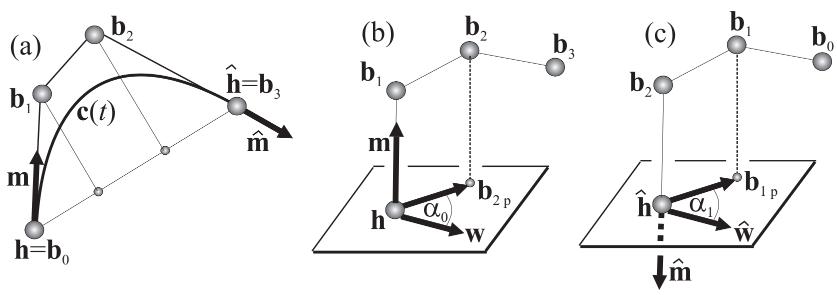


Figure 4.11: (a) constructing the cubic Bezier curve $\mathbf{c}(t)$ to move $(\mathbf{h},\mathbf{m},\mathbf{w})$ to $(\widehat{\mathbf{h}},\widehat{\mathbf{m}},\widehat{\mathbf{w}})$; (b)-(c) computing α_0 and α_1 for constructing $\alpha(t)$.

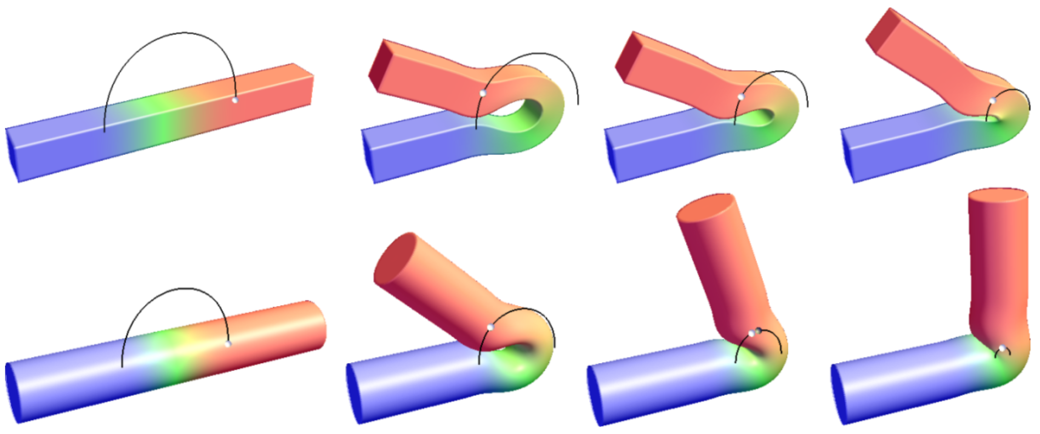


Figure 4.12: Our approach allows for rather strong bending deformations. The deformation behavior can be controlled by adjusting the radius of the curve.

Figure 4.11a gives an illustration. Figure 4.12 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 to $\mathbf{m}$. Figure 4.11b 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 4.11c 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 $(\widehat{\mathbf{h}},\mathbf{m},\mathbf{w})$,

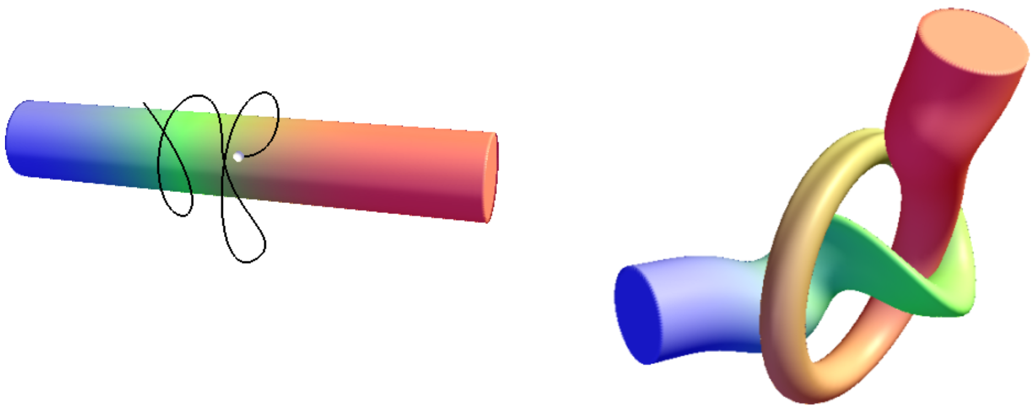


Figure 4.13: Thanks to the surface-based definition of $r(\mathbf{x},t)$ and the curve-guided deformation, extreme deformations like knots are possible.

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

$$e(\mathbf{x},t) = \mathbf{t}\cdot(\mathbf{x}-\mathbf{z}) \quad , \quad f(\mathbf{x},t) = (\mathbf{t}\times(\mathbf{x}-\mathbf{z}))^2. \tag{4.13}$$

4.4.3 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 doesn’t steer the inner deformation by a handle but by a curve and a twisting function, we can handle such deformations. Figure 4.13 shows an example of making a ”knot” into a cylinder model. Figure 4.14 shows a box which is twisted and at the same time deformed along a curve.

4.5 Evaluation

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

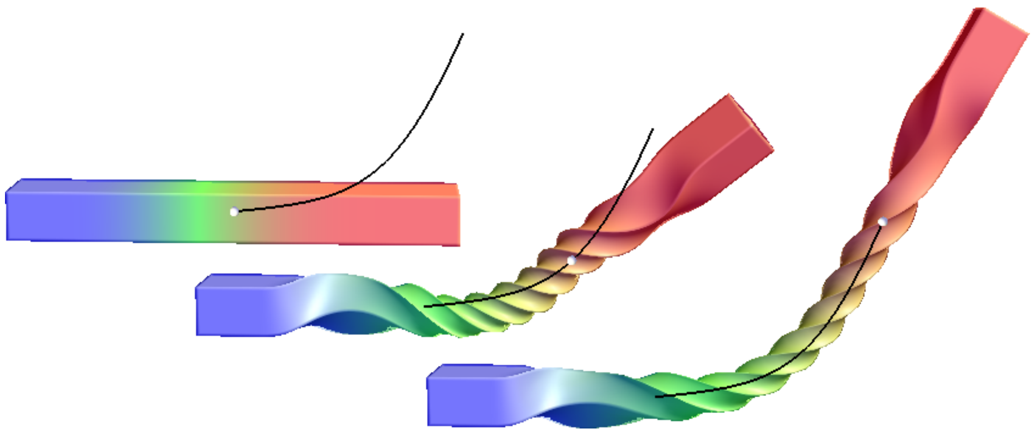


Figure 4.14: Also extreme twistings combined with a deformation along the control curve are possible.

model	fig.	vertices	t/step $[ms]$	vol. error
box	4.3	2,462	4	1.8%
bar	4.12	9,730	9	0.4%
squirrel	4.5	9,995	9	0.08%
horse	4.8	19,851	22	0.7%
cylinder	4.13	21,302	20	1.6%
hand	4.6	53,054	55	0.01%

Table 4.1: The benchmark shows that the GPU implementation performs interactively and the volume change is very low.

4.5.1 Performance and volume-preservation

In Table 4.1, we see a performance benchmark of our GPU implementation, performed on a GeForce 7800 GTX graphics card. In addition, we measured the change of volume with respect to the undeformed shape. 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.

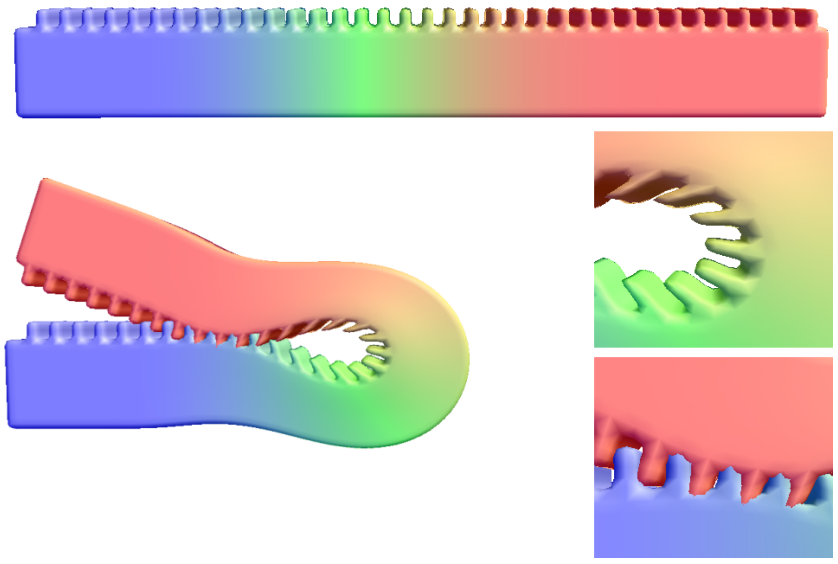


Figure 4.15: Our deformations tend to prevent local self-intersections, while global self-intersections are possible.

4.5.2 Self-intersections

The shape deformation technique from Chapter 2 guarantees to prevent local and global self-intersections because the computation is based on globally defined 3D continuous scalar fields. Since the method in this chapter computes r locally on the shape surface, 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 4.15 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 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 4.16 shows

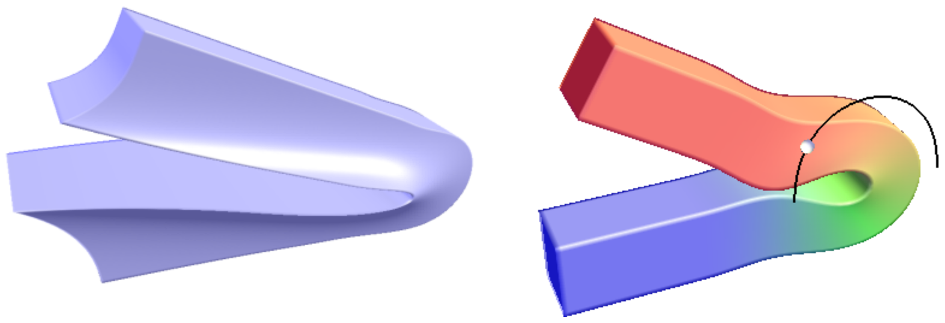


Figure 4.16: While the original vector-field-based shape deformation method (left) prevents global self-intersections, it introduces strong distortions under extreme deformations. Our surface-based method (right) tolerates global self-intersections, with the advantage that extreme deformations are free of distortions and the amount of deformation can be exactly controlled.

a comparison of a strong bending deformation using the method from Chapter 2 and the method from this chapter. 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.

4.6 Discussion

4.6.1 Contributions

We presented an approach to explicitly control vector-field-based shape deformations. In the following, we list the most relevant contributions:

Exact control: In contrast to the original approach presented in Chapter 2, where the region of influence was controlled by simple implicit objects like spheres or cylinders, our new 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 it on the GPU. The main difference to the original method described in Chapter 2 is that our implementation does not need any read-backs from GPU to CPU during the deformation.

4.6.2 Inherited features

By utilizing the vector-field-based shape deformation technique, our approach inherits the following features:

Intuitive editing: By utilizing divergence-free vector fields, 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.

4.6.3 Limitations

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

Global self-intersections: Contrary to the method from Chapter 2, our method cannot guarantee to avoid global self-intersections.

Vertex dependencies: The method in Chapter 2 is able to integrate the vertices of the mesh independently of each other. Contrary to this, the surface-based approach needs the connectivity information of the mesh to estimate the volumetric field r out of the surface field s.

5 Elastic Secondary Deformations by Vector Field Integration

As demonstrated in the previous chapters, the vector-field-based shape deformation method tends to produce physically plausible results. This is due to the fact that the way it preserves volume gives the impression of working with real incompressible material. This property makes the method also attractive for animation: For instance, the soft tissue of human beings or animals is (nearly) incompressible. In this chapter, we explore applications of vector-field-based shape deformations to model this kind of animation.

For the treatment of many applications, a deformation or animation can be divided into a primary and a secondary structure [OZH00]. A primary animation/deformation performs rather large and global changes of the shape. Typical examples are keyframe animations, interactive movement of solids, rigid body simulations including collision detection, or the interactive deformation of shapes. In addition to this, secondary deformations perform rather small changes of the shapes but contribute significantly to the realism of the scene. Secondary deformations can be explicitly modeled (e.g. facial animations or lip synchronization in animated full-body human characters) or they can be derived from the primary animation/deformation by assuming certain elastic material properties of the shape. An example of secondary deformations are jiggling and bouncing effects on the moving skin on a human body [PH06].

In the following, we present an approach to model elastic secondary deformations by constructing and integrating time-dependent divergence-free vector fields. Elastic deformations are connected to certain intrinsic forces and tend to move back to the original shape if these forces cease. This way, jiggling and oscillating effects can be represented. The deformation described here is steered by incorporating basic mechanical laws coming from the primary animation/deformation. In fact, a low number of mass-spring sets is simulated to steer the deformation. The results of this simulation are used for an on-the-fly construction of time-dependent vector fields, delivering the new position of each vertex by a numerical path line integration. It guarantees that the deformations are volume-preserving, and without (local or global) self-intersections.

To model secondary deformations which are derived from a primary motion, two general approaches are possible. Firstly, physically based approaches consider the inherent structure of the shape to simulate the deformations. Secondly, heuristic approaches describe the deformation by a low number of parameters in order to obtain plausible and fast deformations without the explicit consideration of physical laws. The technique presented here can be considered as a compromise between the two general approaches. While the deformation is described by a simple heuristic model (a lower number of mass-spring sets), their kinematic simulation is exact as well as the deformation itself is guaranteed to preserve important physical properties (volume, avoidance of self-intersections). In particular, it turns out that these properties are strong enough to yield plausible deformations if the mass-spring sets are placed in an appropriate way.

The rest of the chapter is organized as follows: Section 5.1 reviews related work and Section 5.2 describes our vector-field-based approach. Section 5.3 describes how our model can handle collisions in the primary animations. Section 5.4 describes details of our GPU based implementation. Section 5.5 applies our technique to different approaches for the primary animations: interactive moving of solid bodies, keyframe animations, rigid body simulations, and interactive deformations. We evaluate and compare our method in Section 5.6 and conclusions are drawn in Section 5.7.

5.1 Related work

There is a huge body of approaches to define animations and deformations. Here we only mention approaches which explicitly deal with secondary deformation/motion, or which can incorporate secondary motion effects. In general, all approaches aim in finding appropriate combinations of physically exact simulations and simplifying assumptions to get plausible deformations at interactive rates.

Mass-spring systems are an intuitive technique to deform objects realistically. Initially used for facial modeling [PB81], other fields like skin, fat and muscle simulation [CHP89, TW90, TW91, NT98] and interactive animation of structured deformable objects [DSB99] have been addressed using mass-spring systems. In order to simulate larger mass-spring systems in real-time, [GEW05] developed a GPU implementation which simulates spring elongation and compression on the graphics card and renders the deformed surface.

The Finite Difference Method has been introduced by [TPBF87] as a tool to simulate elastically deformable models. The Finite Element Method has been used to simulate elastic [DDCB01, GKS02, MDM$^+$02] and elastoplastic fracturing [OBH02, MG04] material. The method has also been used to obtain interactive skeleton-driven deformations [CGC$^+$02, ZG05] and physically based rigging of deformable characters [CBC$^+$05]. By employing modal analysis, which reduces the computational complexity of such simulations remarkably, [JP02, CK05] were able to obtain complex deformations at ineractive rates on the GPU. [TBHF03] used the Finite Volume Method to simulate skeletal muscle. [JP99] proposed the Boundary Element Method for simulating deformable objects accurately and interactively.

The above-mentioned approaches have in common that they rely on connected structures like mass-spring networks, grids or volumetric meshes. In contrast to this, mesh free methods [DC96, Ton98] abandon connectivity. Point based animation allows for animation of elastic, plastic and melting objects [MKN$^+$04]. Here, both the particles on which the simulation is carried out and the object surface are represented by points without connectivity.

Trying to strictly adhere the laws of physics in order to get realistic deformations often comes at the cost of performance. This makes it difficult to simulate scenarios of moderate complexity in real-time. Non-physically motivated approaches sacrifice realism for performance. [MHTG05] proposed a geometrically motivated model for simulating deformable objects. This mesh free approach replaces energies by geometric constraints and forces by distances between current and goal positions. This way, the dynamic simulation is efficient and unconditionally stable.

5.2 Our approach

The main idea of our approach is to control the secondary deformation by a low number of parameters which we describe as a number of mass-spring sets (Section 5.2.1). For each of the mass-spring sets we define a local deformation which is based in the integration of a divergence-free vector field (Section 5.2.2). Based on this, different ways of composing them are possible (Section 5.2.3). Also, the impact of a particular deformation can be limited to certain parts of the shape (Section 5.2.4). Section 5.2.5 describes how to place and parametrize the mass-spring sets. Section 5.2.6 describes how a level-of-detail approach can be incorporated into our concept.

5.2.1 Mass-spring sets

Mass-spring systems are popular and well-understood tools to simulate various phenomena, among them deformations. Here we use a very simple one-degree-of-freedom mass-spring system and call it a *mass-spring set* (this naming should reflect that this is indeed one of the simplest mass-spring systems one can imagine). It consists of two points $\mathbf{p}, \mathbf{q}$ which are connected by a spring with stiffness k and damping parameter c. The spring has a length of zero, i.e. $\mathbf{p}$ and $\mathbf{q}$ coincide in the rest state. While $\mathbf{p}$ is considered as a fixed anchor point inside a shape, $\mathbf{q}$ is equipped with a certain mass $m_{\mathbf{q}}$. Then the position of $\mathbf{q}$ can be simulated

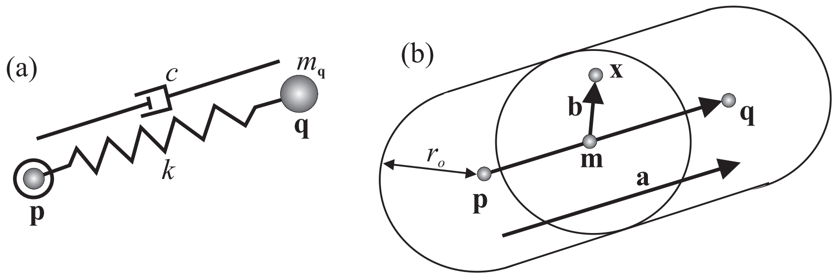


Figure 5.1: (a) A mass-spring set: $\mathbf{p}$ is a fixed point inside the shape, $\mathbf{q}$ is a freely moving point, equipped with a mass $m_{\mathbf{q}}$; (b) configuration to define deformation $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ and vector field $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$.

by basic rules of mechanics if the shape (and therefore $\mathbf{p}$ inside it) undergoes a primary motion. In fact, our simulation includes spring damping, inertia, and the repeated conversion of potential and kinetic energy. See the corresponding section on mass-spring systems in [NMK$^+$06] for an explanation of these methods. Figure 5.1a illustrates a mass-spring set.

5.2.2 Constructing the deformation

For a mass-spring set with the points $\mathbf{p},\mathbf{q}$ at a certain location, we define a space deformation $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} : \mathbb{R}^3 \to \mathbb{R}^3$ with the following properties:

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{p}) = \mathbf{q} \tag{5.1}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}) = \mathbf{x} \quad \text{for} \quad dist(\mathbf{x},\overline{\mathbf{p}\,\mathbf{q}}) > r_o \tag{5.2}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \;\text{ is } C^2 \text{ continuous} \tag{5.3}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \;\text{ is volume preserving} \tag{5.4}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \;\text{ does not produce self-intersections} \tag{5.5}$$

where $dist(\mathbf{x},\overline{\mathbf{p}\,\mathbf{q}})$ describes the minimal Euclidean distance between a point $\mathbf{x}$ and the line segment $\overline{\mathbf{p}\,\mathbf{q}}$. The positive value r_o controls the area of impact of the deformation. The main idea to get $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ is to construct a divergence-free time-dependent vector field $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t)$ and obtain $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ as the result of a path line integration of $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$. To do so, we define the auxiliary vector field

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) = \gamma(\alpha\,\mathbf{a} + \beta\,\mathbf{b}) \tag{5.6}$$

with

$$\begin{aligned} \mathbf{a} = \mathbf{q} - \mathbf{p} \quad , \quad \mathbf{m} = (1-t)\,\mathbf{p} + t\,\mathbf{q} \quad , \quad \mathbf{b} = \mathbf{x} - \mathbf{m} \\ \alpha = r_o^2 - 5\,\mathbf{b}^2 \quad , \quad \beta = 4\,\mathbf{a}\mathbf{b} \quad , \quad \gamma = \frac{\left(r_o^2 - \mathbf{b}^2\right)^3}{(r_o^2)^4}. \end{aligned} \tag{5.7}$$

Figure 5.1b illustrates this. Then it is a straightforward exercise in algebra to show that $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ has the following properties:

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}\left((1-t)\,\mathbf{p} + t\,\mathbf{q}\;,\,t\right) = \mathbf{q} - \mathbf{p} \tag{5.8}$$

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_3 \quad \text{for} \quad r_o^2 = \mathbf{b}^2 \tag{5.9}$$

$$\nabla\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_{3,3} \quad , \quad \nabla\nabla\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_{3,3,3} \quad \text{for} \quad r_o^2 = \mathbf{b}^2 \tag{5.10}$$

$$\operatorname{div}(\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}) \equiv 0 \tag{5.11}$$

where $\mathbf{0}_3$, $\mathbf{0}_{3,3}$, $\mathbf{0}_{3,3,3}$ are the 3D zero tensors of order $1,2,3$, respectively. (5.9) and (5.10) mean that for $r_o^2 = \mathbf{b}^2$, $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ and its first and second order partials vanish. Now we can define $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ as

$$\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) = \begin{cases} \mathbf{w}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) & \text{for} \quad \mathbf{b}^2 \le r_o^2 \\ \mathbf{0}_3 & \text{else} \end{cases} \tag{5.12}$$

and $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ as the result of a path line integration over the time interval $[0,1]$:

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}) = \mathbf{x} + \int_0^1 \mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}(s),s)\,ds. \tag{5.13}$$

Then (5.9), (5.10) and (5.12) give that $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ is C^2 continuous, which implies (5.3). (5.8) yields (5.1). (5.2) follows from the piecewise definition of $\mathbf{v}_{\mathbf{p},\mathbf{q},r_0}$ in (5.12). (5.4) and (5.5) follow directly from (5.11) [vFTS06]. Note that $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ is a piecewise polynomial vector field of degree 8. Figure 5.2 illustrates $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},0)$, $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},\frac{1}{2})$, $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},1)$ for $r_o = \frac{1}{2}\|\mathbf{q}-\mathbf{p}\|$ using illuminated stream lines [ZSH96].

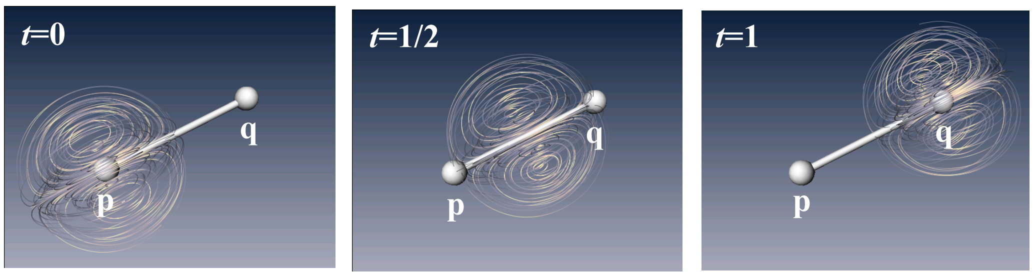


Figure 5.2: $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t)$ with $t = 0, \frac{1}{2}, 1$ for $r_o = \frac{1}{2}\|\mathbf{q}-\mathbf{p}\|$.

Note that the vector field $\mathbf{v}$ constructed by (2.2) – (2.5) defined in Chapter 2 is a piecewise C^1 continuous vector field of degree 16 (if r describes the the Euclidean distance to a center point). The vector field $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ defined by (5.6)–(5.12) appears to be significantly simpler and smoother: C^2 continuous and of degree 8. This enhancement is achieved by letting the inner ring of the deformation collapse to a point, making the expensive (in terms of polynomial degree) C^1 joint between inner and intermediate region unnecessary. In fact, $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ was constructed similar to [vFTS06] as

$$r(\mathbf{x}) = (\mathbf{x}-\mathbf{m})^2 \quad , \quad r_i = 0 \tag{5.14}$$

and $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \|\mathbf{q}-\mathbf{p}\| \cdot (\nabla p \times \nabla q)$ with $p = (1-b)\cdot e + b\cdot 0$ and $q = (1-b)\cdot f + b\cdot 0$ where e, f are linear scalar fields with $(\nabla e)^2 = (\nabla f)^2 = 1$, $\nabla e \cdot \nabla f = \nabla e \cdot \mathbf{a} = \nabla f \cdot \mathbf{a} = 0$, and $e(\mathbf{m}) = f(\mathbf{m}) = 0$. Since this way the inner region vanishes, the blending function can be simplified to

$$b(r) = \sum_{i=0}^{2} w_i\, B_i^2\left(\frac{r-r_i}{r_o-r_i}\right). \tag{5.15}$$

with $(w_0, \ldots, w_2) = (0, 1, 1)$. This explains the reduction in the polynomial degree in $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$. To explain the improved continuity, we note that $\mathbf{v}$ constructed by (2.2)–(2.5) is actually C^1 at the joint between inner and intermediate region, and it is C^2 between intermediate and outer region. Since we make the inner region disappear, the remaining global continuity is C^2.

5.2.3 Composition of vector field integration

In general, more than one mass-spring set is independently used to get the desired elastic deformation. Since any linear combination or time-concatenation of divergence-free vector fields is divergence-free as well, there are two simple options to compose two or more mass-spring sets. Let two mass spring sets be given by $\mathbf{p}_1, \mathbf{q}_1, r_{o1}$ and $\mathbf{p}_2, \mathbf{q}_2, r_{o2}$ respectively. Then we use the following compositions:

1. Adding the vector fields: the deformation is obtained by a path line integration of $\mathbf{v}(\mathbf{x},t) = \mathbf{v}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}} + \mathbf{v}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}$ over the time interval $[0,1]$.
2. Concatenating the vector field: here
$$\mathbf{v}(\mathbf{x},t) = \begin{cases} \mathbf{v}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}}(\mathbf{x},t) & \text{for } \ 0 \le t < 1 \\ \mathbf{v}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}(\mathbf{x},t-1) & \text{for } \ 1 \le t \le 2 \end{cases}$$
is integrated over the time interval $[0,2]$.

Note that concatenating the vector fields corresponds to a concatenation of $\mathbf{d}_{\mathbf{p}_i,\mathbf{q}_i,r_{oi}}$: the integration of $\mathbf{v}(\mathbf{x},t)$ leads to $\mathbf{d}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}(\mathbf{d}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}}(\mathbf{x}))$.

In order to deform smaller details of the shape correctly, they are deformed by first using an addition of vector fields with small influences. Then the shape is further deformed by concatenating an addition of vector fields with larger influences. Figure 5.3 illustrates this and Section 5.4.1 gives more details about our implementation of this approach.

5.2.4 Deformation skinning

Being a space deformation, the described deformation technique may not produce desirable results if independent parts of the shape are spatially close to each other. For instance, suppose a mass-spring set is placed in the right foot of the camel model (Figure 5.4 left). When the spring is elongated, the left leg can enter the influence region and will be deformed as well. In order to prevent such situations, we need a mechanism to constrain the deformation to a specific segment of the shape. We use an extension to a standard technique called matrix palette skinning

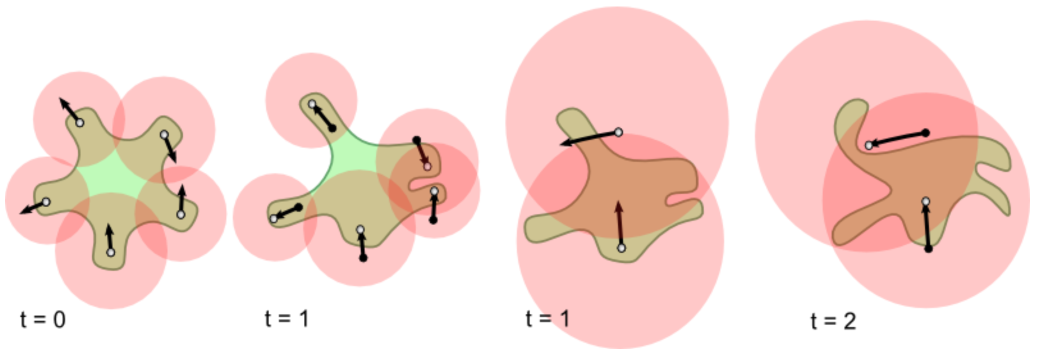


Figure 5.3: For $0 \leq t < 1$, small details are deformed using an addition of vector fields. Afterward, for $1 \leq t \leq 2$, larger areas are deformed using vector fields with larger influence.

(indexed skinning), which is both GPU-friendly and straightforward to implement [LKM01]. It works by assigning a set of usually four index-weight pairs to each vertex. Each index points to an element of a matrix palette, which is an array of 4x4 matrices defining affine transformations. A vertex is deformed by computing its transformations for all four indices and computing the weighted sum of them. Usually, an animated character is segmented into its different body parts, and a transformation is assigned to each segment. The weight for this transformation is 1 for most vertices of the segment – only at the joints, where two or more segments meet, the weights are chosen such that a smooth blending between the different transformations is achieved. Usually the weights are specified together with a skeleton (which defines the affine transformations of the segments) by an animator in a 3D authoring tool. We extend this by additionally assigning a list of mass-spring sets to each segment. Each list of mass-spring sets defines a composed vector field. This means that instead of a palette of matrices, we have a palette of matrices plus vector fields. The extended skinning algorithm works basically the same as before: the vertex is deformed by transforming it using matrix multiplication and by afterward deforming it using vector field integration, for each of the four palette indices. The resulting vertex is again the weighted sum of these four deformed positions. Section 5.4.2 describes the implementation in more detail. Figure 5.4 shows how a mass-spring set deforms a leg of the camel model without affecting other body parts by using deformation skinning.

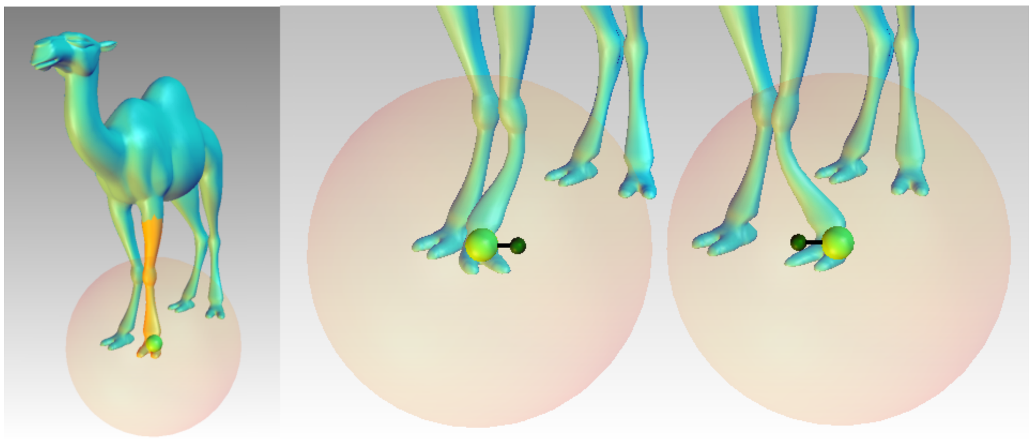


Figure 5.4: Using deformation skinning, the deformation described by the mass-spring set can be constrained to one leg of the camel.

Note that this surface-based skinning does not prevent global self-intersections and does not exactly preserve volume at the parts where different deformations are blended. However, global self-intersections can be avoided by performing an additional collision detection between shape segments. The areas where multiple deformations are blended (e.g. at the joints of a character) are usually small, so the effect of non-exact volume-preservation is negligible.

5.2.5 Setting the mass-spring sets

The physical plausibility of our approach strongly depends on number, location and parametrization of the mass-spring sets. Since it turns out that a rather low number of mass-spring sets already gives pleasing results and that the location of ”good” sets follows the shape intuitively, we left the placing of the mass-spring sets to the user. However, in our experiments we realized a number of rules of thumbs to get pleasing secondary deformations.

Mass-spring sets should be placed close to the center of large homogeneous areas. If only one mass-spring set is used, it should be placed close to the center of gravity of the object. Furthermore, the area of influence should approximately reflect the size of the object. In addition, further mass-spring sets with smaller

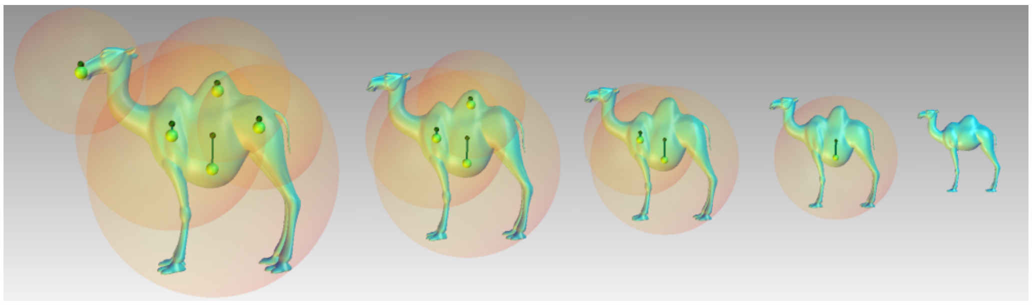


Figure 5.5: Level of detail: with increasing distance, more and more mass-spring sets can be omitted.

area of influence can be placed into distant parts of the shape like head, legs or ears. Furthermore, $m_{\mathbf{q}}, k, c$ have to be set for each mass-spring set. We have chosen $m_{\mathbf{q}}$ proportional to the area of influence while k is constant for all springs. That way, mass-spring sets with larger influence move more slowly than others, resulting in a composition of motions with different frequencies.

5.2.6 Level of detail

Motivated by the fact that mass-spring sets which are far away from the viewer and/or have a small influence radius are visually not significant, we can take advantage of a simple level of detail technique in order to increase performance. Let $\mathbf{p}$ be the anchor point and r_o be the radius of influence of a mass-spring set. Given the distance d between $\mathbf{p}$ and the viewer, we use this mass-spring for deformation only if $r_o > t \cdot d$, where t is a user defined threshold. That way, only mass-spring sets that are close enough and whose influence is large enough are considered during deformation. Figure 5.5 illustrates this.

5.3 Collision handling

Realistic secondary deformations should be able to react properly on collisions of the shape coming from the primary animation. In this section we show that our approach does so by simply concatenating an additional vector field to the integration. This enables us to combine our technique with a rigid body simulation. The idea is to allow objects to penetrate during the rigid body simulation and adding a repelling force which is proportional to the depth of penetration. In order to remove the penetration, we deform the soft object by integrating a new concatenated vector field $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$. This way we can simulate elastic collisions.

5.3.1 Definition of $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}$

The construction of $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}$ follows Chapter 2 in the following way: A center point $\mathbf{c}$ should be translated along a direction vector $\mathbf{d}$ to $\mathbf{c}+\mathbf{d}$ by integrating over a time interval of 1; r_i and r_o denote the inner and outer radius of the deformation. We apply (2.2)–(2.5) with the choices $r(\mathbf{x},t) = \| \mathbf{c}+t\,\mathbf{d}-\mathbf{x} \|$ and $e(\mathbf{x},t)$, $f(\mathbf{x},t)$ are linear scalar fields with $(\nabla e)^2 = (\nabla f)^2 = 1$, $\nabla e \cdot \nabla f = \nabla e \cdot \mathbf{d} = \nabla f \cdot \mathbf{d} = 0$, and $e(\mathbf{c}+t\,\mathbf{d},t) = f(\mathbf{c}+t\,\mathbf{d},t) = 0$. Then we get $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t) = \|\mathbf{d}\| \cdot (\nabla p(\mathbf{x},t) \times \nabla q(\mathbf{x},t))$.

5.3.2 Placing the vector field

After defining $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$, we need to place it such that it deforms the shape realistically under collision. More precisely, we have to choose $\mathbf{c}$, $\mathbf{d}$, r_i, r_o such that interpenetrations between the elastic body and the collision geometry are canceled. We assume that the collision geometry (i.e. the objects that the body can collide with) is decomposed into convex hulls. While automatic methods exist [LA04], we decomposed our scenes manually. As described in Section 5.2.4, the mesh of the elastic object may be decomposed into different segments. From now on, we treat each segment independently as an *elastic body*. Let B be an elastic body penetrating a convex part C of the collision geometry. Furthermore let $\mathbf{c}_B$ be the center of gravity of B and $\mathbf{c}_C$ be the center of gravity of C. Then, for r_i we choose the maximum distance between $\mathbf{c}_C$ and all points in C, i.e. $r_i = \max\{\|\mathbf{x}-\mathbf{c}_C\| : \mathbf{x} \in C\}$. That way, we make sure that C lies completely in the inner region with the result – as we will see later – that no point of B ever enters C. The parameter r_o basically determines the (visual) softness of the material: the smaller r_o, the softer appears

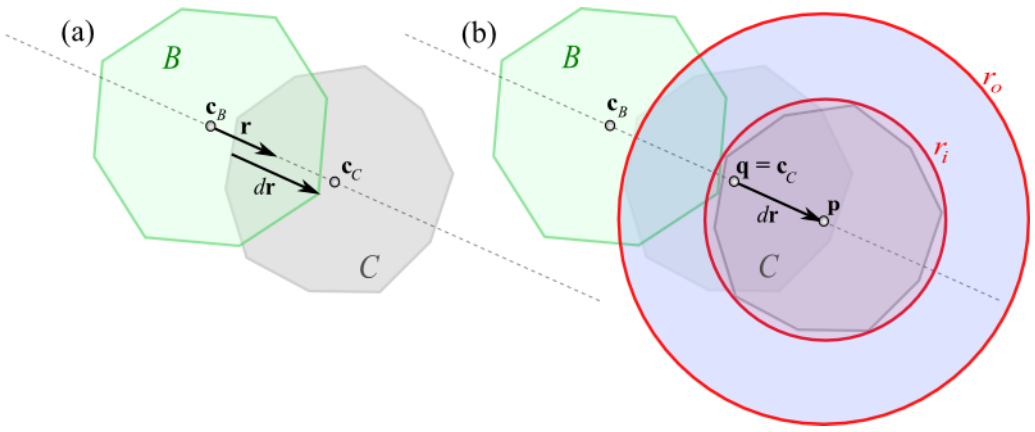


Figure 5.6: (a) Body B penetrates collision geometry C with penetration depth d. (b) The vector field is placed by setting parameters $\mathbf{p}, \mathbf{q}, r_i, r_o$ appropriately.

the material. We found that setting $r_o - r_i$ proportional to the size of the body gives pleasing results. To do that, we use the maximum distance of all points of B from $\mathbf{c}_B$ as a measure for the size of B and compute $r_o = r_i + s \cdot \max\{\|\mathbf{x} - \mathbf{c}_B\| : \mathbf{x} \in B\}$, where s is a user defined softness factor. In our tests we used values between 1 and 2. Of course, r_i and r_o can be precomputed for better performance.

To simplify matters, we allow only translations along the vector $\mathbf{c}_B - \mathbf{c}_C$. In order to determine $\mathbf{c}$ and $\mathbf{d}$, we first compute the maximum penetration depth d of B in C along the vector $\mathbf{r} = \frac{\mathbf{c}_C - \mathbf{c}_B}{\|\mathbf{c}_C - \mathbf{c}_B\|}$. Figure 5.6 illustrates this. Then we can set
$\mathbf{c} = \mathbf{c}_C + d\mathbf{r}$
and $\mathbf{d} = \mathbf{c}_C - \mathbf{c}$. If we now center C at $\mathbf{c}$ instead of $\mathbf{c}_C$, C and B would touch each other but not interpenetrate. Furthermore, since all points of C are completely in the inner region, where we have a constant vector field, no point of B can ever enter C during integration. Altogether, the deformation defined by $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$ cancels the penetration of B into C by pushing the penetrating parts of B out of C.

In the case of multiple collisions, for each convex collision geometry C that is penetrated by B, we compute the corresponding vector field $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$ as described above. These fields are summed up and finally concatenated with the vector field computed from the mass-spring sets (Section 5.2.2).

5.4 Implementation

The presented deformation technique is totally independent of any mesh connectivity information or control lattices like grids or volumetric meshes. It is based on an integration of vector fields defined by a relatively small number of mass-spring sets. This fact allows for a direct GPU implementation of the shape deformation. While the simulation of the mass-spring movements is still carried out on the CPU, the resulting spring positions are sent to the GPU which is responsible for deforming and rendering the mesh. Since the deformation always starts from the original, undeformed mesh, the original mesh can be stored as a static vertex buffer (together with buffers for normals, indices, weights etc.) in video memory, i.e. the vertex positions don't have to be updated. Practically this means that the mesh is deformed by rendering it. This is a contrast to the GPU implementation in Chapter 2, which relies on a read-back of vertex positions from video memory, which is a performance bottleneck.

5.4.1 Composition

As mentioned in Section 5.2.3, we can compose vector fields in two ways: either by addition or by concatenation. In order to get the best results, we use a combination of both. We construct three vector fields $\mathbf{v}_{local}, \mathbf{v}_{global}, \mathbf{v}_{collision}$ and concatenate them. $\mathbf{v}_{local}$ is the summation of all vector fields defined by mass-spring sets whose radius is smaller than some user defined threshold. This means that $\mathbf{v}_{local}$ gives a rather local deformation. $\mathbf{v}_{global}$ is a summation of all vector fields defined by mass-spring sets whose radius is larger than the threshold. That way, it deforms the shape more globally. Finally $\mathbf{v}_{collision}$ is constructed as described in Section 5.3 and concatenated to the other fields. Figure 5.7 illustrates this.

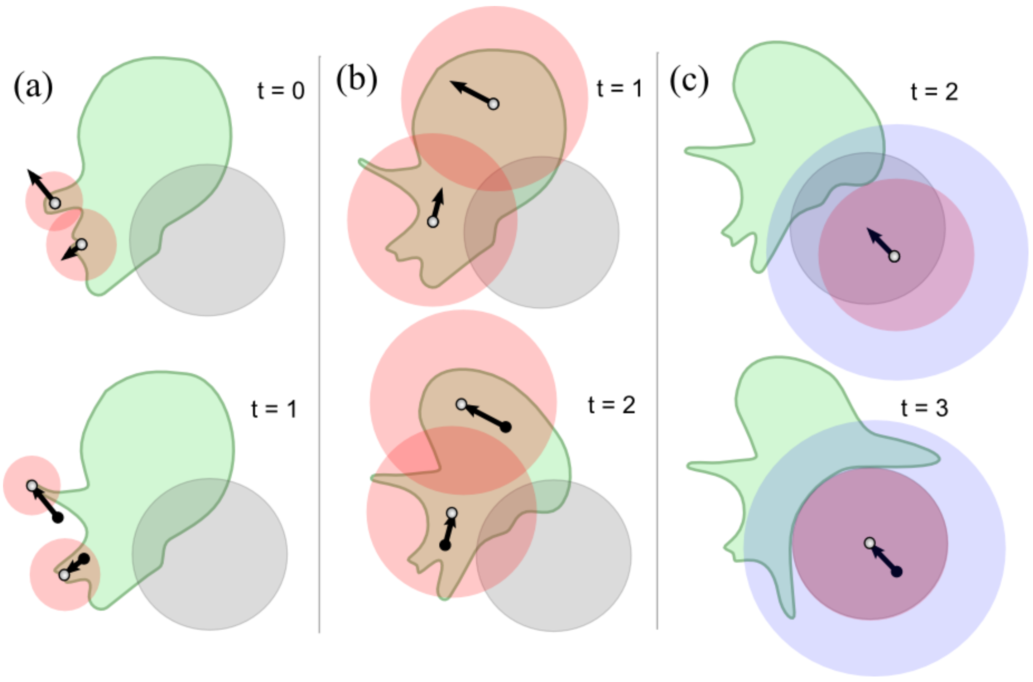


Figure 5.7: (a) A body (green) is deformed locally using $\mathbf{v}_{local}$. (b) Afterward, it is deformed more globally by $\mathbf{v}_{global}$ which is defined by mass-spring sets with larger influence. (c) Finally the penetration into the collision object (grey) is reversed using $\mathbf{v}_{collision}$.

5.4.2 Skinning and integration

As explained in Section 5.2.4, we combine the vector field integration with matrix palette skinning in order to constrain the deformation to specific segments and to be able to emulate elasticity for animated objects and characters. A vertex $\mathbf{x}$ is deformed using the following formula:

$$\mathbf{x}' = \sum_{k=1}^{4} w_k \mathbf{d}_{i_k}(\mathbf{M}_{i_k}\mathbf{x}). \tag{5.16}$$

Here, i_k, w_k are the index-weight pairs for this vertex, $\mathbf{M}_{i_k}$ is the i_kth matrix from the matrix palette, and $\mathbf{d}_{i_k}(.)$ is the i_kth deformation from the vector field palette.

The algorithm can be optimized by skipping index-weight pairs whose weight is zero. The numerical integration needed for $\mathbf{d}_{i_k}$ is carried out using a standard Euler integration of the composed vector fields from Section 5.4.1. It turns out that even a small number of integration steps gives pleasing results. In our implementation we used twelve steps.

5.4.3 Normal computation

In order to get a correct lighting of the deformed shape, we need to compute the deformed normals as well. Since we want to avoid read-backs from the GPU, we cannot employ standard methods considering for instance the 1-ring neighborhood of a vertex. Our solution works as follows: Instead of deforming only the vertex $\mathbf{x}$, two additional points $\mathbf{x}_1, \mathbf{x}_2$ in the tangent plane of $\mathbf{x}$ and close to $\mathbf{x}$ are deformed using the same indices and weights as $\mathbf{x}$. Then the deformed normal can be computed as the normalized of $(\mathbf{x}'_2 - \mathbf{x}') \times (\mathbf{x}'_1 - \mathbf{x}')$, where $\mathbf{x}', \mathbf{x}'_1, \mathbf{x}'_2$ are the deformed positions of $\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2$ respectively.

5.4.4 GPU implementation

The necessary parameters like spring positions, elongations and bone matrices are passed to the vertex shader as uniform variables. As mentioned above, the vertices, indices, weights, normals etc. are stored as static buffers in video memory. In order to prevent vertices from being deformed multiple times, we exploit the vertex cache by computing cache-optimized triangle strips of our models. For that purpose, we used the NvTriStrip library[1].

5.5 Applications

In this section we apply our technique to different kinds of primary animations.

[1] *http://developer.nvidia.com/object/nvtristrip_library.html*

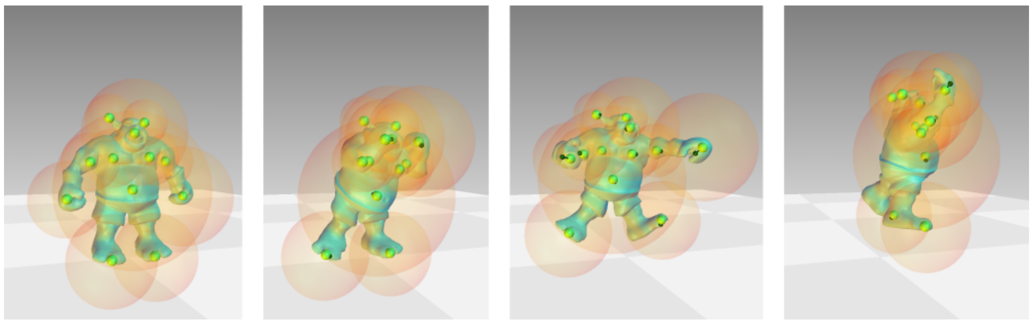


Figure 5.8: Skeletal animations become more lifelike by adding secondary animations of muscles and fat.

5.5.1 Keyframe animation

In interactive applications like games, character animation is usually performed via skeletal animation. That means that for every part of the body a rigid transformation is defined to get different poses of the character. Being rigid, these transformations cannot describe elastic secondary deformations resulting from jiggling muscles or fat. In order to demonstrate our method, we have built a simple keyframe animation system based on a linear interpolation between different character poses. Figure 5.8 shows the animation of a boxer with and without our secondary deformations. Here, the animator has placed 13 masses with different influences in the character. During the animation, the corresponding mass-spring sets are simulated based on the movement of the body parts. Then the shape is deformed and rendered by the GPU (see Section 5.4). The secondary deformations introduced by vector field integration enhance the visual appearance and make the animation more lifelike.

5.5.2 Interactive moving and deformation

Even simple operations like moving or rotating objects as well as applying standard deformation techniques can be visually enhanced by adding elastic secondary deformations to the object. In Figure 5.9 the cow model is deformed and moved interactively. In addition, the mass-spring sets which are placed throughout the body are simulated based on the resulting motion. By choosing different influ-

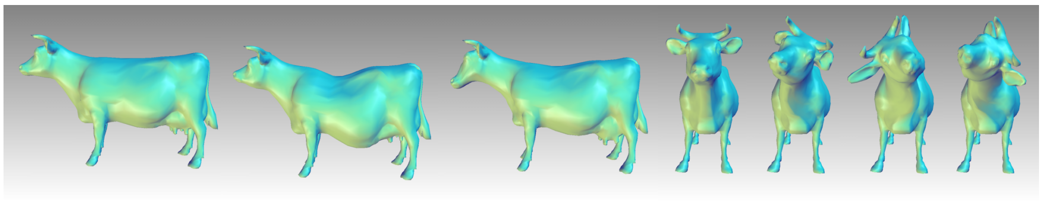


Figure 5.9: During interactive movement and deformation of the cow model, different parts of the body jiggle elastically with different frequencies.

ence radii, masses and spring parameters, realistic looking motions of fat and muscle can be emulated. Usually, mass-spring sets with low influence and mass are placed at small details (like the ears of the cow model) to get fast vibrations there, while large and plump regions (like the belly of the cow model) are deformed by mass-spring sets with large mass and influence.

5.5.3 Rigid body simulation

In order to get realistically moving and colliding elastic bodies, we apply our secondary deformation to rigid body simulation methods. That means that an object is represented by one or more rigid bodies which are interconnected by joints. These rigid bodies are simulated using a standard rigid body dynamics system. For our implementation, we used the *Newton Game Dynamics*[2] library. Using deformation skinning (Section 5.2.4), we get a ragdoll simulation of the object. In order to make the object appear more elastic, each rigid body is rotated towards its rest pose during the simulation. This basically means that the bodies are interconnected by elastic joints. Furthermore, we have to allow the object to penetrate the collision environment, such that we can apply the collision handling algorithm from Section 5.3. During penetration, we apply an impulse to the object which is proportional to the penetration depth. That way, we get an elastic collision.

Figure 5.10 shows a scene of a camel model falling through a scene composed of walls and bars. The model is composed of 7 different rigid bodies for legs, body, neck and head, and 5 mass-spring sets are distributed in the model. In the

[2] *http://www.newtondynamics.com*

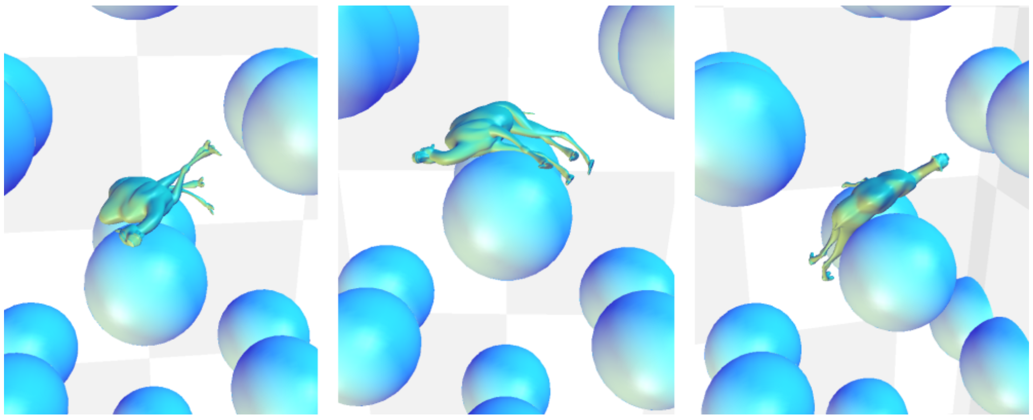


Figure 5.10: A camel ragdoll falls through a tower with obstacles.

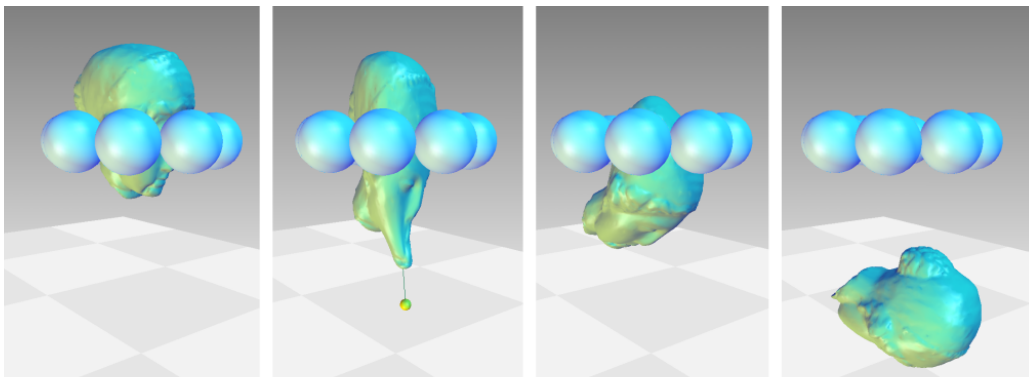


Figure 5.11: While the model is pulled through the ring, its volume is preserved and intersections with the ring geometry are avoided.

scene shown in Figure 5.11, a head model simulated by one rigid body and five mass-spring sets is dragged through a ring. Here we can see how the volume of the shape is preserved and intersections with the collision geometry are avoided. Figure 5.12 shows how the shape is deformed by multiple draggers. Each dragger contributes a new vector field to the integration.

5.6 Evaluation and Comparison

In order to get a formal evaluation of the approach, physically exact ground truth deformations of real models are necessary. Since our approach is heuristic in

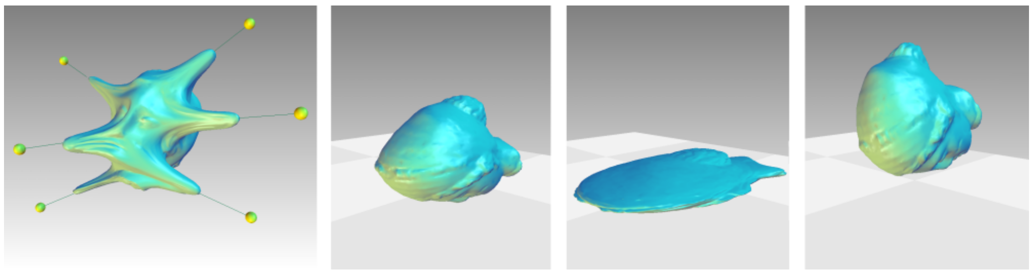


Figure 5.12: Using vector field integration, multiple draggers can be used to deform the model. Even after extreme deformations, the model returns to its original shape.

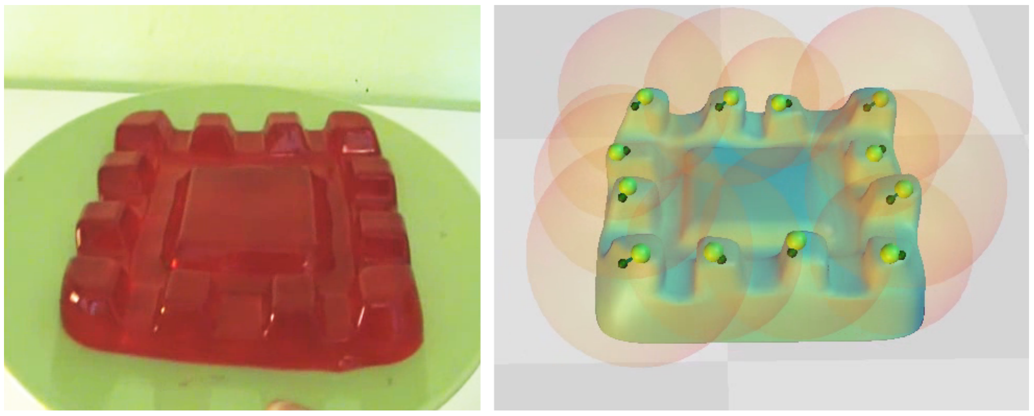


Figure 5.13: In order to evaluate the visual plausibility of our approach, we compared a real jelly (left) with a virtual jelly whose secondary motion is emulated by our system (right).

nature, we prefer to use the visual plausibility as one parameter of our evaluation. The examples show that the secondary deformations look rather realistic even though only a low number of steering parameters are used. In order to get a comparison between a real elastic incompressible material and our method, we recorded a video of a jelly. In Figure 5.13 we see a real jelly which is shaked in order to obtain inertial motion. As a result, the small knobs on the jelly jiggle with different frequencies. We modeled this jelly using our approach, which is shown in Figure 5.13. Here we placed mass-spring sets with different masses and influence radii in the knobs, which took us less than one minute. When the virtual jelly is shaked in our system, the resulting deformation looks quite similar to the real one.

Fig.	*#t*	*#s*	*#m*	*fps*
5.9	5,804	9	8	374 – 376
5.8	15,596	8	13	133 – 135
5.10	19,536	7	5	101 – 125
5.11	16,532	1	4	55 – 135
5.9	5,804	9	8	302 – 355
5.14	345,944	7	10	10 – 14

Table 5.1: Performance benchmark for several models. *#t* is the number of triangles, *#s* the number of skinning segments, *#m* the number of mass-spring sets and *fps* the number of frames per second (worst and best).

Another point of evaluation is the choice and parametrization of the mass-spring sets. Without doing a formal user study, we presented the system to different people (mainly students). It turned out that even when using the system for the first time, it took them only a few minutes of ”playing around” with the mass-spring sets to get realistic secondary deformations similar to the ones shown in the figures.

Furthermore, a number of ”hard” evaluations are possible concerning the following properties: due to its nature, our deformations are volume preserving, C^2 continuous for mass-spring simulations (but only C^1 if collision treatment is involved), and without any self-intersections. We believe that these conditions are strong enough to produce visually plausible deformations even though no explicit physical model is involved. Thanks to the avoidance of complex physical simulations, the method is stable. Figure 5.12 shows a shape that undergoes an extreme deformation during collision and still returns to its rest state afterward.

Due to the fast vector field integration and the GPU implementation, the algorithm allows to emulate high resolution models at interactive rates. For instance, the model shown in Figure 5.14, consisting of 345,944 triangles, 7 segments and 10 mass-spring sets, can be deformed and simulated at 10-14 frames per second. Table 5.1 shows detailed benchmark results for different scenes, measured on an 2.6 GHz CPU with a GeForce 7800 GTX graphics card.

Comparison to existing approaches.

In order to evaluate our contributions, we point out the most important differences

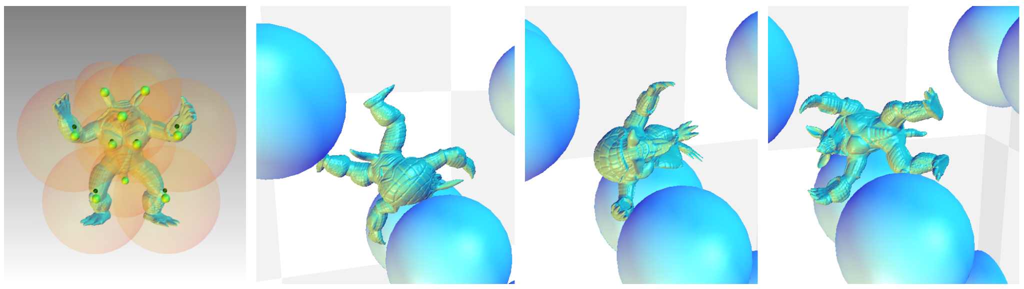


Figure 5.14: Also high-resolution models consisting of several hundred thousands of triangles can be deformed in real-time.

to previous approaches of deformable object modeling.

In contrast to mass-spring systems, our mass-spring sets are not interconnected. This makes the simulation straightforward to implement, intuitive, fast and stable. Furthermore, since the actual deformation is performed by an integration of divergence-free vector fields, the volume of the shape is preserved and a low number of mass-spring sets suffices to give plausible deformations.

While Finite Difference Methods, Finite Element Methods and Finite Volume Methods require control structures like grids or volumetric meshes, our approach is mesh-free. This allows for a fast and intuitive placement of mass-spring sets and requires no preprocessing.

Existing mesh-free methods like point based animation usually require a larger amount of particles in the shape than our approach needs mass-spring sets. This is because of the fact that the underlying physical simulation requires a certain particle density in order to be accurate. In contrast to this, our method gives plausible results even for a small number of mass-spring sets, which is due to its inherent volume-preserving nature.

The geometrically motivated approach by [MHTG05] resembles our approach in the sense that it is mesh-free, requires a small number of particles and exchanges physical accuracy for interactivity and stability. However, this approach does not preserve the volume of the shape, which is especially noticeable for large deformations. While this approach needs to embed the shape into regularly placed cubical regions to get more detailed deformations, our method requires mass-spring sets with appropriate influence radii and positions.

Limitations.
Like most related approaches, the presented work has some limitations. We did not consider collisions between multiple elastic bodies. Here, a convex decomposition of the colliding shapes and a similar algorithm as presented in Section 5.3 might work out. Furthermore, the vector fields described by the mass-spring sets perform only translations in their inner regions. That way, bending or twisting deformations are not directly possible. Here, rotational fields as presented in Chapter 2 might be a solution. Also a resampling of the deformed mesh is not applicable to our approach, because it is GPU based without read-backs. Although highly controllable compared to related approaches, steering the deformation exactly is difficult due to the simple, implicitly defined influence of the mass-spring sets and their corresponding vector fields. Using deformation skinning, prevention of global self-intersections and exact preservation of volume are not guaranteed, as depicted in Figure 5.4. However, we believe that deformation skinning adds to the visual plausiblity and furthermore integrates well with existing skeleton-based animation frameworks.

5.7 Discussion

In this chapter we made the following contributions:

- We introduced the construction and integration of divergence-free vector fields to get elastic secondary deformations.
- In comparison to Chapter 2, we enhanced the used vector fields both in polynomial degree and continuity.
- Contrary to Chapter 2, the GPU implementation does not perform any read-back operations during the integration. This allows real-time deformations even for fairly large shapes.
- We have shown that vector field integration can also be used to avoid unwanted intersections and penetrations of soft objects in rigid body deforma-

tions. It turns out that the combination of rigid body simulation as primary animation and our secondary deformation based on vector field integration gives a realistic emulation of elastically deforming models. Furthermore, it can be built on top of an existing rigid body system, which makes the implementation more unified and robust.

Our deformations are stable, smooth, and, without deformation skinning, volume preserving and free of self-intersections. They allow to apply LOD approaches. Furthermore, they can be used on top of arbitrary primary animations.

6 Volume-preserving Mesh Skinning

Mesh skinning is a standard technique which is widely used in Computer Graphics, especially in the context of character animation. The idea is to bind a mesh to a skeleton whose joints can be transformed in order to obtain a smooth non-rigid deformation of the surrounding mesh. The deformation of each mesh vertex is computed as a weighted blend of the joint transformations. While the method is intuitive and fast, producing convincing and physically plausible deformations can be quite challenging. Common problems are unnatural volume changes in deforming joint regions – examples are the well known ‘collapsing joint’ and ‘candy wrapper’ effects.

We present a technique which automatically preserves the volume for arbitrary skeletal mesh deformations. The volume-preservation can be adjusted manually by defining weights on the mesh surface. That way, the animator can precisely control how the volume is preserved to get more realistic deformations. The presented method has a number of advantages over existing volume-preserving deformation approaches:

- The volume preservation is exact.
- The volume correction is obtained directly by a closed-form solution.
- Even during strong articulations the volume is preserved without distortion artifacts.

- No additional volumetric structures like control meshes, tetrahedral meshes or implicit deformers are required.

- The animator has precise control over the volume preservation. It is possible to define per-vertex how the volume should be preserved.

6.1 Approach

Let us consider a triangle mesh with vertex positions $\mathbf{P} = [\mathbf{p}_1, ..., \mathbf{p}_n]$ and a triangulation $T \subseteq \{1, ..n\}^3$. In this notation, T contains for each triangle a triple of vertex indices. If the surface is a closed manifold, we can compute the volume of the shape as the sum of the signed volumes of the tetrahedra formed by each triangle and the origin:

$$\text{volume}(\mathbf{P}) = \frac{1}{6} \sum_{(i,j,k) \in T} \mathbf{p}_i \cdot (\mathbf{p}_j \times \mathbf{p}_k) \tag{6.1}$$

In above formula '$\cdot$' denotes the scalar product and '$\times$' the cross product. Given a deformed mesh with vertex positions $\mathbf{P}' = [\mathbf{p}'_1, ..., \mathbf{p}'_n]$ and the same triangulation, we want to modify $\mathbf{P}'$ so that the volume of the deformed mesh is the same as the volume of the undeformed mesh. To do so, we define a displacment field $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_n]$ which tells for each vertex position $\mathbf{p}'_i$ in which direction and how strong it has to be moved so that the original volume is reached. The displacement field can be freely defined as long as it changes the volume of the shape. The exact scaling of $\mathbf{V}$ can be computed automatically, as we will see below. Formally, we want the following condition to hold:

$$\text{volume}(\mathbf{P}' + \lambda \cdot \mathbf{V}) = \text{volume}(\mathbf{P}) \tag{6.2}$$

λ is the factor by which $\mathbf{V}$ needs to be scaled. The remaining task is to find a λ which fulfills Equation 6.2. By insertion and reordering we get

$$\begin{aligned}\sum_{(i,j,k)\in T} (\mathbf{p}'_i + \lambda \mathbf{v}_i)((\mathbf{p}'_j + \lambda \mathbf{v}_j) \times (\mathbf{p}'_k + \lambda \mathbf{v}_k)) \\ -\mathbf{p}_i(\mathbf{p}_j \times \mathbf{p}_k) = 0\end{aligned}$$

$$\Leftrightarrow c_0 + \lambda c_1 + \lambda^2 c_2 + \lambda^3 c_3 = 0 \tag{6.3}$$

with

$$\begin{aligned}
c_0 &= \sum_{(i,j,k)\in T} \mathbf{p}'_i(\mathbf{p}'_j \times \mathbf{p}'_k) - \mathbf{p}_i(\mathbf{p}_j \times \mathbf{p}_k),\\
c_1 &= \sum_{(i,j,k)\in T} \mathbf{p}'_i(\mathbf{p}'_j \times \mathbf{v}_k) + \mathbf{p}'_i(\mathbf{v}_j \times \mathbf{p}'_k)\\
&\qquad + \mathbf{v}_i(\mathbf{p}'_j \times \mathbf{p}'_k),\\
c_2 &= \sum_{(i,j,k)\in T} \mathbf{p}'_i(\mathbf{v}_j \times \mathbf{v}_k) + \mathbf{v}_i(\mathbf{p}'_j \times \mathbf{v}_k)\\
&\qquad + \mathbf{v}_i(\mathbf{v}_j \times \mathbf{p}'_k),\\
c_3 &= \sum_{(i,j,k)\in T} \mathbf{v}_i(\mathbf{v}_j \times \mathbf{v}_k).
\end{aligned}$$

The cubic equation (6.3) has up to three real solutions λ[1]. Since we want to change $\mathbf{P}'$ as little as possible, we choose the solution whose absolute value is minimal.

In a nutshell, we can preserve the volume of arbitrary deformations in the following way:

1. Deform the mesh vertices $\mathbf{P}$ to get $\mathbf{P}'$.
2. Define a displacement field $\mathbf{V}$.
3. Compute scaling factor λ.
4. Set the final vertex positions to $\mathbf{P}' + \lambda \mathbf{V}$.

[1]We use Cardano's method for solving cubic equations.

In the following section we will see how this approach can be applied to skeleton-based mesh skinning. We propose a method to automatically define **V** for arbitrary skeletal deformations and how to adjust **V** manually to get more complex volume-preserving deformations.

6.2 Volume-preserving mesh skinning

Standard mesh skinning usually works by binding a skeleton to a mesh where the influence of each joint has varying weights across the mesh vertices. The weights are set by the animator during the rigging process. The mesh is deformed by blending joint transformations with respect to these weights. Given k joints with transformation matrices $\mathbf{M}_1,...,\mathbf{M}_k \in \mathbb{R}^{4\times4}$ and vertex weights $W_1,...,W_k$ with $W_i = [w_{i,1},...,w_{i,n}]$, let the weights be defined such that they sum up to one for each vertex, i.e. $\sum_{i=1}^{k} w_{i,j} = 1$ for $j = 1,...,n$. Each deformed vertex position is computed as a linear combination of the transformed bind positions:

$$\mathbf{P}' = \sum_{i}^{k} W_i \cdot \mathbf{M}_i(\mathbf{P}) \tag{6.4}$$

Here, '$\cdot$' denotes the componentwise scalar multiplication and $\mathbf{M}_i(\mathbf{P})$ is the transformation of all points $\mathbf{P}$ by joint matrix $\mathbf{M}_i$.

Since our volume preservation is global, we need to carry out the deformations at each joint one after another. We do this in hierarchical order, i.e. we start at the root joint and traverse recursively along the children. For each joint deformation, we apply our volume correction algorithm and thus can perserve the volume locally at the corresponding joint. After each deformation, $\mathbf{P}$ is replaced by $\mathbf{P}'$. To simplify matters, we consider in the following only one deformation at one joint, i.e. we assume that $\mathbf{P}$ and the bone transformations $\mathbf{M}_i$ are initialized properly.

First, we construct an initial displacement field **U** in bind pose which will be used to construct **V** for each joint deformation. Given start position $\mathbf{a}_i$ and end position

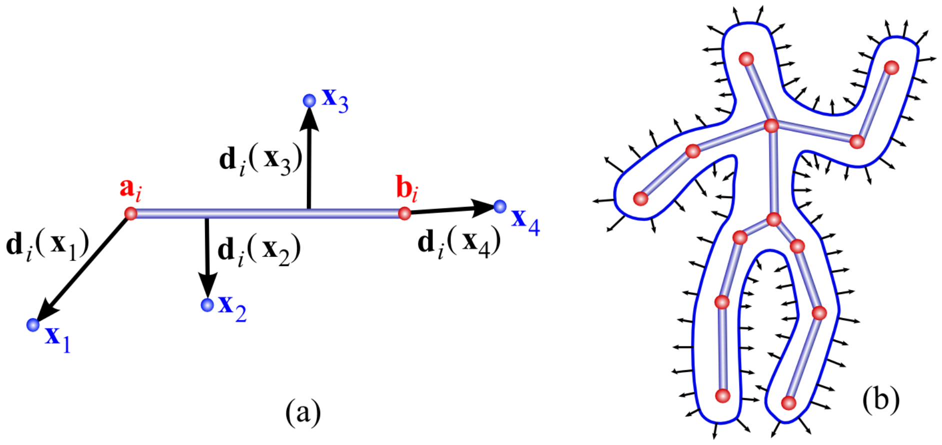

$\mathbf{x}_3$
$\mathbf{d}_i(\mathbf{x}_3)$
$\mathbf{a}_i$
$\mathbf{b}_i$
$\mathbf{x}_4$
$\mathbf{d}_i(\mathbf{x}_1)$
$\mathbf{d}_i(\mathbf{x}_2)$
$\mathbf{d}_i(\mathbf{x}_4)$
$\mathbf{x}_2$
$\mathbf{x}_1$
(a)
(b)

Figure 6.1: (a) The vector field $\mathbf{d}_i$ evaluated at a number of sample points $\mathbf{x}_j$. (b) We obtain $\mathbf{U}$ by blending the fields $\mathbf{d}_i$ at the mesh vertices.

$\mathbf{b}_i$ of each bone in bind pose, we can define a 3D vector field

$$\mathbf{d}_i(\mathbf{x}) = \mathbf{x} - (\mathbf{a}_i + \mathrm{clamp}(\frac{(\mathbf{b}_i - \mathbf{a}_i)\cdot(\mathbf{x}-\mathbf{a}_i)}{\|\mathbf{b}_i - \mathbf{a}_i\|^2})(\mathbf{b}_i - \mathbf{a}_i)) \tag{6.5}$$

with

$$\mathrm{clamp}(\delta) = \begin{cases} 0 & \text{if} \quad \delta < 0 \\ 1 & \text{if} \quad \delta > 1 \\ \delta & \text{else.} \end{cases} \tag{6.6}$$

Visually, $\mathbf{d}_i(\mathbf{x})$ is the vector from the closest point on bone segment $(\mathbf{a}_i, \mathbf{b}_i)$ to $\mathbf{x}$, as illustrated in Figure 6.1a.

Now we define $\mathbf{U}$ as the blended combination of all fields $\mathbf{d}_i$:

$$\mathbf{U} = \sum_i^k W_i \cdot \mathbf{d}_i(\mathbf{P}) \tag{6.7}$$

That way, we get a smooth vector field pointing away from the skeleton (Figure 6.1b), which is ideal for our volume correction algorithm. We can now define the deformed field

$$\mathbf{U}' = \sum_i^k W_i \cdot (\mathbf{M}_i(\mathbf{P}+\mathbf{U}) - \mathbf{M}_i(\mathbf{P})) \tag{6.8}$$

analogously to the skinning equation (6.4)[2]. Finally, the displacment vector field for the current deformation is defined as

$$\mathbf{V} = S \cdot \mathbf{U}' \tag{6.9}$$

where $S = [s_1, ...s_n]$ defines a weighing factor at each vertex which basically describes how much the volume correction should affect this vertex. As we will see in Section 6.3, S can be defined manually on the vertices to get complex, user defined deformations. However, there is also an automatic method to generate useful weights at a joint. Given the current joint j and its parent joint i, we simply set $S = W_i \cdot W_j$. In most cases, S has large weights near the joint and decreasing weights away from the joint. That way, $\mathbf{V}$ is only non-zero close to the joint and thus the volume is corrected locally in that region.

6.3 Applications

Using the automatic displacement field construction described in the previous chapter, it is possible to directly preserve the volume of skinned meshes without user interaction. Figure 6.2 shows an example of a bar-shaped mesh which is skinned using a simple skeleton. Using our method, the volume is uniformly and smoothly preserved around the joint regions which gives the impression of deforming a real incompressible solid.

In some cases, uniform volume preservation is undesirable, e.g. when the arm of a human character is bent, we don't want volume preservation at the hard elbow but rather at the soft muscle and tissue areas near that joint. The animator can model this with our approach by painting the weights S by which $\mathbf{U}'$ is multiplied (see Equation 6.9) interactively on the vertices. He is not restricted to use positive weights, also negative weights can be used to achieve interesting effects like muscle bulging or foldings, as demonstrated in Figure 6.3.

Figure 2.3 demonstrates how this technique can be used to model realistic skin and

[2] $\mathbf{M}_i(\mathbf{P}+\mathbf{U}) - \mathbf{M}_i(\mathbf{P})$ is used to transform $\mathbf{U}$ without translation.

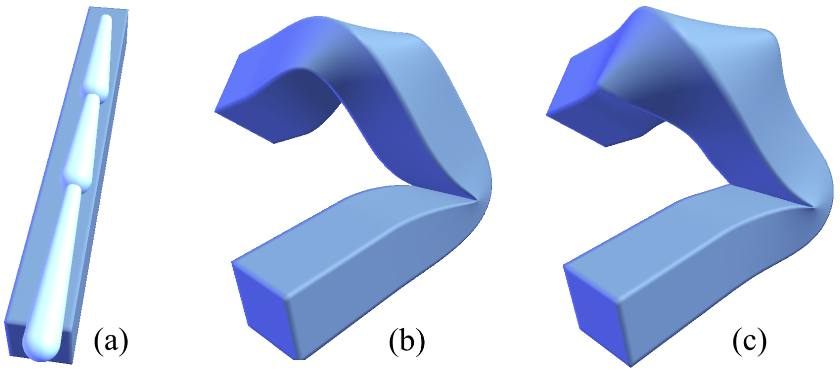


Figure 6.2: (a) A bar is skinned using a skeleton consisting of three bones. (b) The deformed mesh using standard mesh skinning and (c) using volume-preserving mesh skinning.

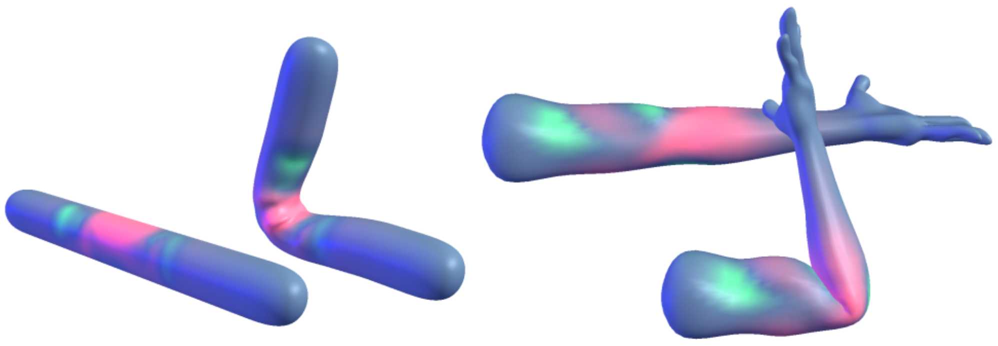


Figure 6.3: The animator can paint positive (pink) or negative (green) weights interactively on the surface. That way, soft tissue deformations and foldings can be modeled.

muscle deformations. Figure 6.4 shows a comparison between traditional mesh skinning and volume-preserving mesh skinning. In Figure 6.4b, where standard mesh skinning was used to twist the arm of a model at the shoulder joint, the ‘candy wrapper’ effect is clearly visible in the shoulder region. This unnatural loss of volume can be prevented with volume-preserving mesh skinning, as shown in Figure 6.4c, where the shoulder region deforms more naturally without volume change.

All models shown in the images were posed in real-time. The deformation of a mesh consisting of 20,000 triangles takes about 28 milliseconds per joint on a

2GHz notebook.

6.4 Discussion

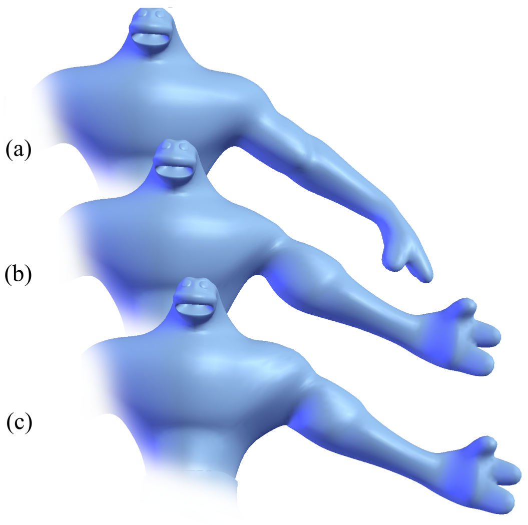


Figure 6.4: (a) A character in bind pose. (b) Using traditional mesh skinning, the twisting deformation produces a ‘candy wrapper’ effect in the shoulder region, while volume-preserving mesh skinning (c) deforms the shoulder naturally.

We introduced a straightforward, but nevertheless effective method to perform mesh skinning with volume-preservation. The key contributions are:

Exact volume preservation: The accuracy of volume preservation often depends on the mesh resolution or the numerical accuracy of the algorithm [AS07, LCOGL07, SHB07]. Our approach can handle arbitrary coarse meshes and does not rely on a numerical approximation, but preserves the volume exactly as a closed-form solution.

Closed-form solution: In contrast to previous volume preservation methods, our approach neither requires non-linear opimizations [HSL$^+$06] nor numerical vector field integration [AS07]. Especially the vector field integration methods have the drawback that large deformations require longer integrations. Our solution is obtained directly, no matter if the deformation is small or large.

Strong articulations: Strong articulations, like bending an arm by more than 90 degrees, are often problematic and lead to distortions near the joint area [vFTS07, AS07]. Instead of turning off volume preservation for such deformations [AS07], our method can handle strong articulations without problem.

Surface-based volume correction: Volume-preserving mesh skinning does not require control meshes [HSL^+06], implicit deformers [AS07] or multiresolution representations [SHB07]. The computation is carried out on the mesh surface and can be controlled on a per-vertex basis: The animator can specify a scalar field on the surface which steers the behavior of the volume correction during deformation. While vector-field based approaches like [AS07] model tissue variation by 3D implicit functions, our approach can handle arbitrary complex variations without performance impact since the variation is defined on the vertices. In fact, the runtime of our algorithm is proportional to the number of vertices times the number of joints that affect the deformation.

Our approach has some limitations compared to existing deformation techniques. In particular, it does not preserve local surface features and does not automatically prevent self-intersections. However, traditional mesh skinning, which is still a standard in character animation, doesn't have these features either: Feature preservation is usually not required for characters and self-intersections resulting from strong articulations are quite common during character posing [BWK03]. For instance, the pose of the rightmost character in Figure 2.3 produces self-intersections which are not noticeable at all. In fact, automatic prevention of self-intersections and volume preservation at the same time during strong articulations is not possible without distortion artifacts up to now [vFTS07, AS07]. We believe that it is a good compromise to keep the volume preservation and leave the prevention of self-intersections up to the animator.

7 Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments

Flow visualization is an active field of research. A variety of techniques for the interactive exploration of flow phenomena has been developed. One approach is to resemble well-accepted techniques from experimental flow visualization. Among them, smoke visualization plays an important role: in a real flow of gases, smoke is advected and its temporal behavior gives information about the flow. Often, the smoke is advected from line structures (e.g., from a burning stick), or it is inserted from certain points (smoke nozzles). Another common technique in experimental flow visualization are wool tufts where small yarns are attached to a body and observed during the flow experiment.

To use smoke in (computer aided) flow visualization, it is represented either in a volumetric, a particle based, or an image based way. All these approaches have proven to be useful. In a volumetric approach, the smoke is represented as a density scalar field, making a high resolution or an adaptive grid necessary to capture certain details. Particle based approaches usually need a high number of particles to represent smoke.

In this thesis we propose an alternative representation of smoke: a semi-transparent streak surface. This approach is inspired by artistic smoke photographs as shown in Figure 7.1. In this image, smoke was advected from a stick, i.e., a line-like seeding structure. Together with appropriate lighting conditions, expressive and aesthetic photographs of real-world smoke were obtained. In Figure 7.1 the smoke clearly forms a semi-transparent surface structure which can be interpreted as a streak surface. Hence, Figure 7.1 makes us believe that semi-transparent streak surfaces can give expressive visual representations of a flow if we follow the smoke metaphor, i.e., if the surface is rendered to look like smoke.

Stream surface integration is a standard approach in flow visualization. However, the integration of *streak* surfaces in time-dependent flows is fundamentally different, because it requires an adaptive remeshing of the *complete* surface at every time step. Up to now, this prevented streak surfaces from being used in interactive applications.

The main idea of our method is to use streak surfaces without any adaptive remeshing, i.e., the triangular mesh representing the streak surface has a fixed topology and connectivity. This will lead to large and non-regular triangles e.g. due to diverging flow, but these areas become less visible because of the optical model for smoke that we apply for the rendering. This way, we combine two advantages: the surface looks like a smoke structure, and no time is spent for surface remeshing.

The smoke surface technique obtained this way can be enhanced in several ways. By coloring the mesh vertices, we can visualize time lines and streak lines within the streak surface. The seeding can also be done starting from all vertices of a surface at the same time, leading to semi-transparent time surfaces. Smoke nozzles can be simulated by setting certain parts of the streak surface invisible. Finally, wool tufts can be mimicked by seeding short and narrow streak surfaces close to the obstacles in the flow.

Section 7.1 gives an overview over related visualization techniques. Section 7.2 describes our approach in detail. Section 7.3 describes modifications and enhancements. Section 7.4 applies the approach to a number of test data sets. Section 7.5 evaluates our approach while conclusions are drawn in Section 7.6.

7.1 Related Work

Smoke is a well-researched subject in both computer graphics and visualization. While several offline techniques for the realistic rendering of smoke exist [KH84, JC98], we restrict the overview here to realtime smoke rendering methods.

Probably the most often used approaches for realtime smoke rendering are based on particles. Here, a (usually large) set of particles is seeded into a flow and advected over time. Surface-particles [vW92] are represented as points with normals and can be rendered efficiently to visualize flow. Another early method to render particle-based volumetric data is texture splatting [CM93]. In recent years, point sprites [KW05] and semi-transparent textured billboards with opacity values proportional to the density value of the particles [Hol03] have become a popular method for particle rendering. In order to reduce artifacts for highly stretched flows, blob particles [SF93, AN05] can be used. They are basically ellipsoid particles that can be stretched and split during animation. By considering the spherical geometry of the particles during fragment processing, spherical billboards [USKS06] eliminate clipping and popping artifacts which occur when the particles flow around other objects in the scene. By using a variant of the depth difference technique, non-photorealistic cartoon rendering of smoke particles is possible [SMC04].

With the increasing power of graphics hardware, realtime volumetric rendering methods of gaseous phenomena are becoming more and more popular. Animated clouds can be rendered realistically using a slice-based volumetric rendering scheme [SSEH03]. By shifting complex computations to the GPU, realtime performance is obtained. By utilizing the latest features of current graphics hardware (rendering to 3D textures, geometry shader, stream out), real-time volumetric smoke, fire and water with fluid dynamics have recently been made possible [TL07, CLT07]. Here, the actual rendering is performed via ray casting on the GPU. Using compensated ray marching [ZRL$^+$08], it is possible to incorporate dynamic environment lighting into interactive smoke rendering.

In computer graphics, meshes have been used in the context of cloud rendering

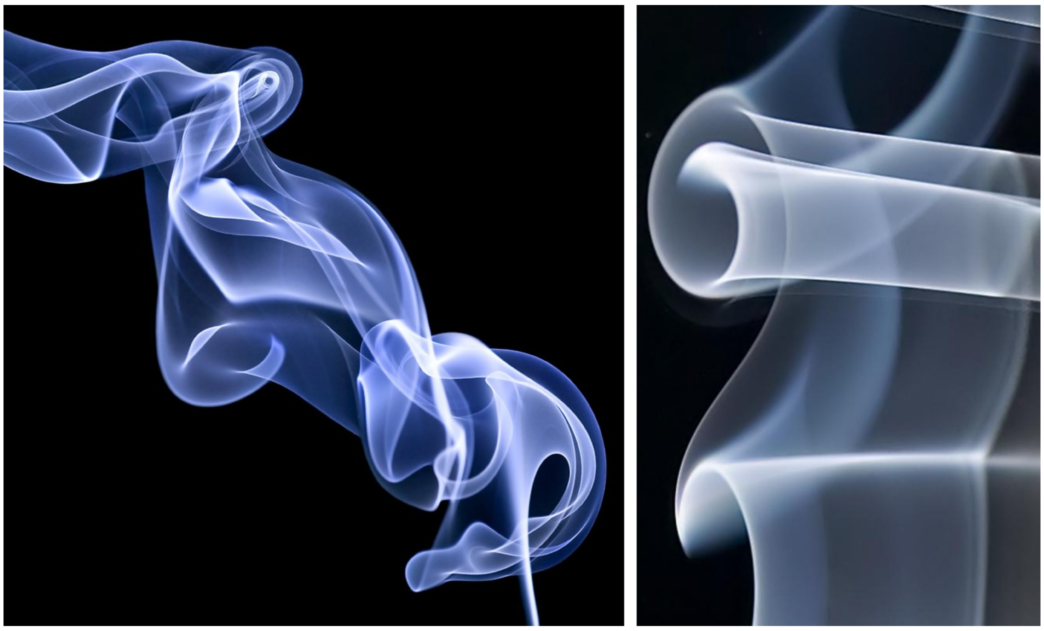


Figure 7.1: Artistic photographs of real smoke. From [Bre05].

[Gar85, TB02, BNL06].

In flow visualization, a number of approaches has been developed to create smoke-like images. Flow volumes [MBC93] were introduced as the volumetric counterpart to stream lines. Using volumetric meshes composed of transparently rendered tetrahedra, the technique allows for interactive exploration of vector fields. Particle based methods [KKKW05, BSK$^+$07] use a large number of particles to get a smoke impression. Image based smoke visualizations are mentioned in [vW02]. As an alternative to smoke, dye has been proposed to visualize vector fields [Wei04]. Virtual tufts for flow visualization are described in [SP02].

7.2 Approach

The main idea of our approach is to represent smoke as a triangular mesh of a fixed topology and connectivity. We use an $(m+1)\times(n+1)$ vertex array $(\mathbf{x}_{i,j}\ i = 0,...,m; j = 0,...,n)$ defining a closed surface of cylinder topology, i.e., we assume $\mathbf{x}_{0,j} = \mathbf{x}_{m,j}$ for $j = 0,..,n$. We call the polygon $(\mathbf{x}_{i,0},..,\mathbf{x}_{i,n})$ the i-th column, while

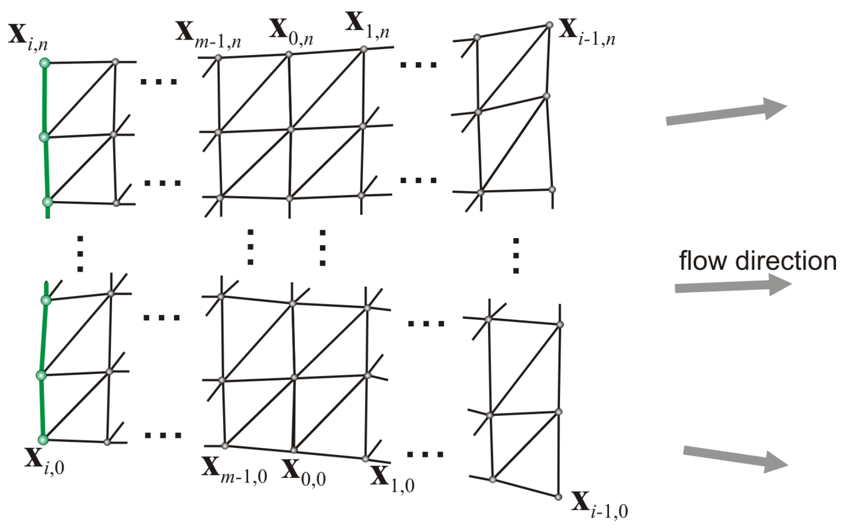


Figure 7.2: Streak surface at time $t_0 + i \cdot \Delta t$.

the polygon $(\mathbf{x}_{0,j}, .., \mathbf{x}_{m,j})$ is the j-th row. As seeding structure we use a polygon $(\mathbf{s}_0, ..., \mathbf{s}_n)$.

For initialization, all vertices are set to the seeding structure, i.e., $\mathbf{x}_{i,j} = \mathbf{s}_j$. Starting at t_0, columns of the array are successively released into the flow. The integration of the i-th column starts at time $t_0 + i \cdot \Delta t$ for $i = 0, ..., m-1$. Note that once a column is released into the flow, it has to be advected in every time step – this is in contrast to stream surfaces, where only the currently last column is integrated. After m time steps, all columns have been consumed at the seeding curve and the complete surface is unfolded into the flow. The seeding continues with the first column, i.e., the vertices of the currently "oldest" column (longest time in the flow) are reset to the seeding structure. This way, a continuous seeding of the streak surface over an unlimited time is possible. Re-allocation is not necessary during the integration: vertices at the end of the streak surface are re-used at the start of it. Figure 7.2 gives an illustration of the streak surface at time $t_0 + i \cdot \Delta t$.

Note that our system allows to change the location of the seeding polygon interactively. Certain optional parts of the opacity computation (explained in the following section) may make it desirable to choose the step width Δt such that the triangles are approximately equilateral shortly after their advection started.

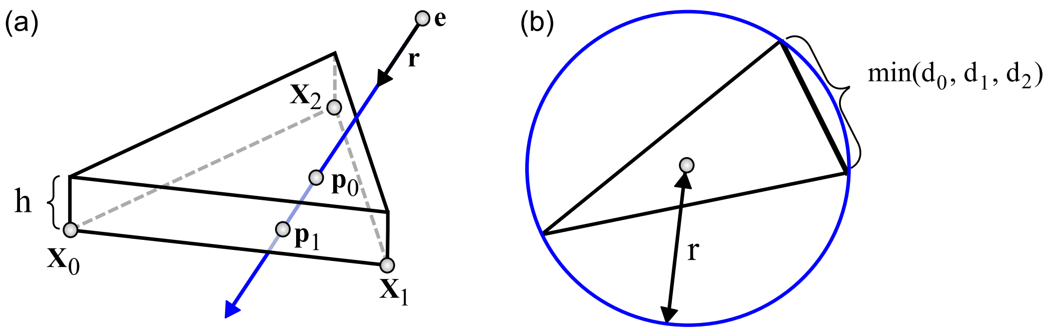


Figure 7.3: Configurations for computing (a) $\alpha_{density}$ and (b) α_{shape}.

Representing a streak surface with a fixed resolution and connectivity seems to be unsuitable since adaptive schemes have already proven their usefulness for much simpler flow features such as stream surfaces. The fixed resolution will lead to situations where e.g. triangles become rather large due to diverging flow behavior. However, the main goal of our approach is not to extract perfect streak surfaces, but to render smoke based on such surfaces. As we will see in the following section, the optical model for smoke already gives that smoke becomes less visible in areas with diverging behavior. In other words, larger triangles are less visible due to the smoke metaphor and therefore the advantages of the fixed resolution (mainly interactivity and ease of implementation) outweigh its shortcomings in our case of smoke rendering.

7.2.1 Opacity Computation

Optical Model of Smoke

To represent the density of smoke, we assume a triangle $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$ to have a certain small height h, i.e., we consider it to be a rather flat prism evenly filled with smoke. Figure 7.3a illustrates the configuration. We assume the viewing ray $\mathbf{p}(t) = \mathbf{e} + t\mathbf{r}$ with $\|\mathbf{r}\| = 1$ entering the prism in the point $\mathbf{p}_0 = \mathbf{e} + t_0\mathbf{r}$ and leaving in $\mathbf{p}_1 = \mathbf{e} + t_1\mathbf{r}$. Then the α value describing the absorption is [Max95]

$$\alpha = 1 - e^{-\int_{t_0}^{t_1} \tau(t)dt} \tag{7.1}$$

where $\tau(t) = \tau(\mathbf{p}(t))$ is the extinction coefficient at the location $\mathbf{p}(t)$ describing the rate that light is occluded. Since we assume τ to be constant inside the prism, (7.1) simplifies to

$$\alpha = 1 - e^{-(t_1 - t_0)\tau}. \tag{7.2}$$

Let γ be the angle between $\mathbf{r}$ and the normal $\mathbf{n}$ of the triangle. Then $(t_1 - t_0) = \frac{h}{\cos\gamma}$ under the assumption that the viewing ray intersects only the spanning triangles of the prism. This gives

$$\alpha(h) = 1 - e^{-\frac{h\,\tau}{\cos\gamma}}. \tag{7.3}$$

Since h is assumed to be rather small, $\alpha(h)$ can be linearized by a Taylor expansion to

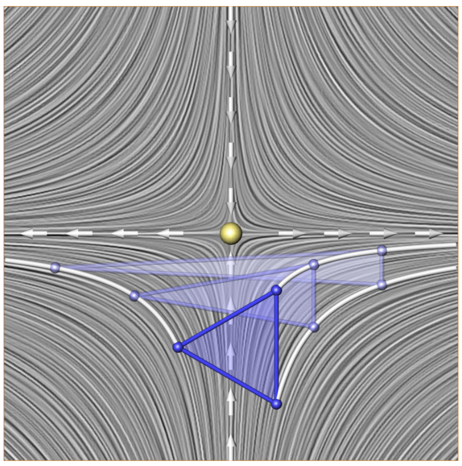
$$\alpha(h) = \alpha(0) + h\,\frac{d\alpha}{dh}(0) = \frac{h\,\tau}{\cos\gamma}. \tag{7.4}$$

Assuming a particle model for the smoke (i.e., the smoke consists of a number of small absorbing particles, τ linearly depends on the particle density inside the prism: $\tau = c\,\frac{n_p}{\text{area}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)}$ where c is a certain constant and n_p is the number of particles within the prism. This yields

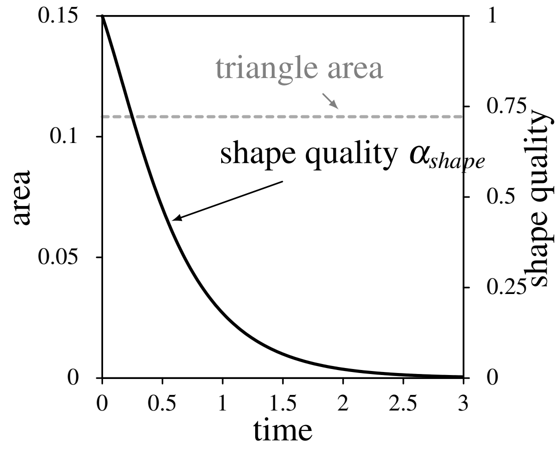
$$\alpha_{density} = \frac{k}{\text{area}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)\,\cos\gamma} \tag{7.5}$$

which describes the α value representing the smoke density inside the prism. The constant $k = h\,c\,n_p$ steers the initial density at seeding time. Note that due to the linearization, $\alpha_{density}$ can be outside the interval $[0,1]$; in this case, it has to be clamped to $[0,1]$.

The physically motivated $\alpha_{density}$ has been used for all smoke visualizations throughout this chapter and it steers the main visual appearance of all these images. However, in a few situations $\alpha_{density}$ does not suffice to compensate for the fixed resolution and connectivity of our mesh. Usually these are small areas where the surface flows around some obstacle or becomes too distorted. The perfect solution in these cases would be to increase the resolution, but this also implies to reduce the speed and responsiveness of the application. If interactive frame rates are desired, we have to trade accuracy for speed eventually. In the following section we have identified some situations leading to visual clutter due to a locally



(a) Integrated triangle. Shown are three instances.



(b) Triangle area and shape quality measure plotted over integration time.

Figure 7.4: Distortion of a equilateral triangle integrated towards the saddle point in the linear vector field $\mathbf{v} = (x, -y)^T$.

too coarse mesh resolution and propose solutions.

Optional Opacity Parameters

In order to detect cuts or other areas where the triangulation does not describe the smoke surface well,[1] we consider a measure of the local quality of the mesh triangles. Note that the area of a triangle does not suffice to detect surface cuts. Figure 7.4a shows an example. There, we have a simple linear vector field containing a saddle point toward which a triangle is integrated while a separation takes place: one vertex moves to the left-hand side, while two go to the right-hand side. This is a typical configuration for a cut surface ending in a long thin triangle which should be set invisible. Note that the area of the triangle is almost constant during the integration (see the dotted line in Figure 7.4b).

A well-accepted measure of the shape quality of a triangle $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2$ is the ratio of the shortest edge length to the radius r of the circumcircle [She02]. The solid line in Figure 7.4b shows the behavior of this measure in the configuration of Figure

[1]Such unsuitable triangles are also the ones connecting the last column of the streak surface with the (freshly reset) first column.

7.4a. Since $r = \frac{d_0 d_1 d_2}{2\,\mathrm{area}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)}$ with $d_0 = \|\mathbf{x}_2 - \mathbf{x}_1\|$, $d_1 = \|\mathbf{x}_0 - \mathbf{x}_2\|$, $d_2 = \|\mathbf{x}_1 - \mathbf{x}_0\|$, we use this to define the shape quality parameter as

$$\alpha_{shape} = \left(\frac{4\,\mathrm{area}(\mathbf{x}_0,\mathbf{x}_1,\mathbf{x}_2)}{\sqrt{3}\,\max\{d_0 d_1\ ,\ d_1 d_2\ ,\ d_2 d_0\}} \right)^s . \tag{7.6}$$

Note that $\alpha_{shape} = 1$ for an equilateral triangle, and that α_{shape} gets smaller the "less equilateral" the triangle is. The positive constant s steers how strong the influence of α_{shape} is relative to $\alpha_{density}$: the smaller s is, the less influential is α_{shape}. For our visualizations s was chosen between 0.5 and 1.

In regions of high Mean curvature of the streak surface, the non-adaptive mesh may not be an appropriate representation. In order to make the surface less visible in this case, we introduce $\alpha_{curvature}$ reflecting the curvature. For a vertex $\mathbf{x}_0$, we compute this as

$$\alpha_{curvature} = 1 - b \cdot \max\{|\mathbf{n}_0\,\mathbf{e}_i| : i = 1..valence(\mathbf{x}_0)\} \tag{7.7}$$

where $\mathbf{n}_0$ is the estimated surface normal at $\mathbf{x}_0$, i iterates over all vertices $\mathbf{x}_i$ in the 1-ring of $\mathbf{x}_0$, and $\mathbf{e}_i = \frac{\mathbf{x}_i - \mathbf{x}_0}{\|\mathbf{x}_i - \mathbf{x}_0\|}$. If all vertices of the 1-ring of $\mathbf{x}_0$ are approximately in the tangent plane of $\mathbf{x}_0$, $\alpha_{curvature}$ is close to 1. The positive constant b determines how strong a large surface curvature influences $\alpha_{curvature}$. We have chosen $b = 2$ and clamp $\alpha_{curvature}$ to the interval $[0,1]$.

Smoke tends to disperse and fade over time. To simulate this behavior, we introduce an additional alpha value α_{fade}. Given the age t of a vertex, which is the time passed after the vertex was seeded, and a maximum age t_{max}, which is the age when a vertex should become invisible, we can compute α_{fade} as

$$\alpha_{fade} = 1 - \frac{t}{t_{max}}. \tag{7.8}$$

We choose t_{max} as the maximum integration time of a vertex before it is seeded again, i.e., $t_{max} = m\Delta t$.

The final α value including all opacity parameters is computed such that it is not larger than its smallest component:

$$\alpha = \alpha_{density}\,\alpha_{shape}\,\alpha_{curvature}\,\alpha_{fade}. \tag{7.9}$$

In practice, we define α per-vertex such that we get a piecewise linear interpolation across the surface. To achieve this, we set $\alpha_{density}$ and α_{shape} of a vertex to the minimum value of its adjacent triangles; $\alpha_{curvature}$ and α_{fade} are already defined per-vertex. $\alpha_{density}$, α_{shape}, $\alpha_{curvature}$ and α_{fade} have to be clamped to the interval $[0,1]$ before applying (7.9).

The final α is steered by three degrees of freedom: k for the initial smoke density, s for the influence of the shape parameter, and b for the influence of the curvature.

Remark: Formulae (7.5) and (7.6) seem to suggest to compute $\alpha_{density}\,\alpha_{shape}$ directly and not separately because $\text{area}(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ cancels out. This fails because $\alpha_{density}$ and α_{shape} have to be clamped separately. Only clamping $(\alpha_{density}\,\alpha_{shape})$ for a triangle with small γ (i.e., close to a silhouette) can make it visible even though it has a bad shape quality.

7.2.2 Implementation

In our OpenGL implementation, we used *depth peeling* [Eve01] in order to avoid depth-sorting of triangles, which would be computationally expensive and can produce artifacts at overlapping triangles. Depth peeling basically works by rendering fragments into different layers which are superimposed afterwards to get the final image. We used four layers in our implementation, which seems to be sufficient for smoke rendering. A big advantage of this method is the fact that we can render the surface as a single triangle strip without changing connectivity.

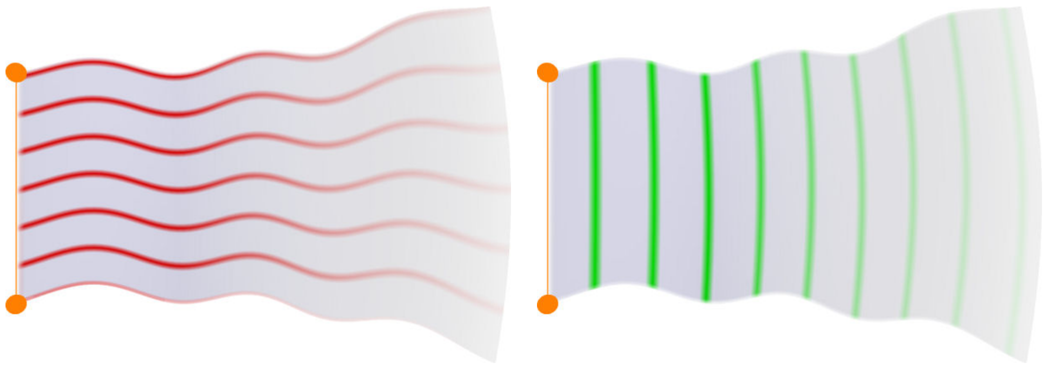
7.3 Enhancements and Modifications

Our approach to rendering smoke surfaces updates the set of vertices by means of advection and computes the alpha values in each frame. Other parts like the connectivity of the surface or rgb-color values remain constant. In this section we propose a number of simple modifications to these constant parts that allow us to achieve a variety of different visualization styles. In particular, we are able to depict streak and time lines, reproduce the visual appearance of smoke nozzles, integrate time surfaces, and mimic an important real-world visualization technique called wool tufts. All these modifications are done in a rather simple preprocessing step. Therefore, they do not influence the rendering performance of our technique.

7.3.1 Streak and Time Lines

A *streak line* is the connection of all particles set out at different times but the same point location. In an experiment, one can observe these structures by constantly releasing dye into the flow from a fixed position. The resulting streak line consists of all particles which have been at this fixed position sometime in the past. Such lines can also be found on our smoke surface under the assumption that the seeding curve remains constant. In this case, the vertices representing a streak line are found in a fixed row of the vertex array: these vertices have been seeded at the same position in space but at different times. Hence, we can depict a streak line by assigning a different color to a row of the array. Figure 7.5a illustrates this.

A *time line* is the connection of all particles set out at the same time but different locations, i.e., a line which gets advected by the flow. An analogon in the real world is a yarn or wire thrown into a river, which gets transported and deformed by the flow. However, in contrast to the yarn, a time line can get shorter and longer. In our implementation, the vertices representing a time line are found in a fixed column of the vertex array: these vertices have been seeded at the same time along the seeding curve. Hence, we can depict a time line by assigning a different color to a column of the array. Figure 7.5b illustrates this.



(a) Coloring fixed rows in the array reveals streak lines.

(b) Coloring fixed columns in the array reveals time lines.

Figure 7.5: Smoke surface in a simple vector field which is constantly seeded at the left seeding line. Streak and time lines can easily be shown on the smoke surface using a simple preprocessing step and at no cost for the rendering performance.

7.3.2 Time Surfaces

Up to now, we seeded particles continuously at a curve – thereby creating a streak surface. We may as well spread out the complete surface in the volume and start the integration of all vertices at once. The surface gets advected and distorted by the flow. In fact, this is a time surface, since all vertices have been seeded at the same time but at different locations. One may think of this as a carpet thrown into a river and transported by the flow.

A problem with this approach is that the surface may rather quickly leave the domain or the visualized smoke simply dissolves after some time. Hence, we have to come up with a scheme for re-injecting smoke. In our implementation, we allow the user to have a small number of time surfaces that can be started and reset interactively.

Time surfaces can aid in understanding the distortion introduced by the flow field. To do so, we color a number of columns and rows in our array such that a uniform grid appears on the surface.[2] Figure 7.6 shows this for the flow behind a circular cylinder (explained in Section 7.4.1). After some integration steps, the grid

[2]Note, that all these grid lines are time lines and *not* streak lines.

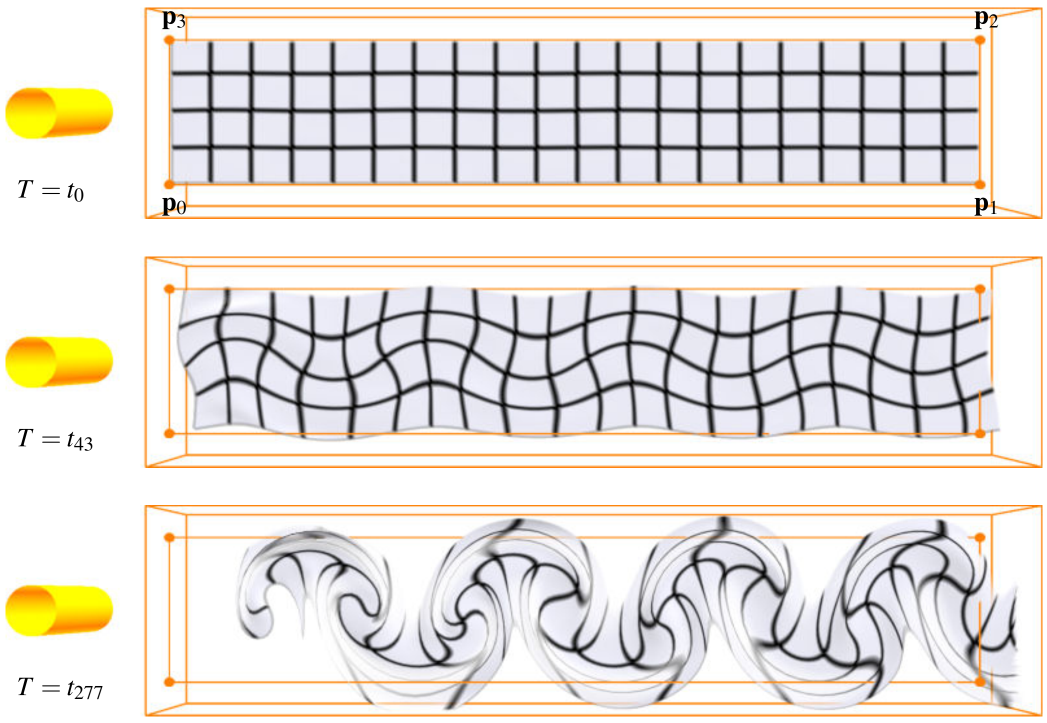
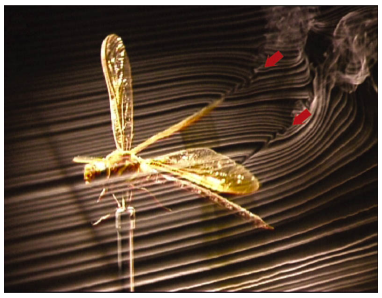
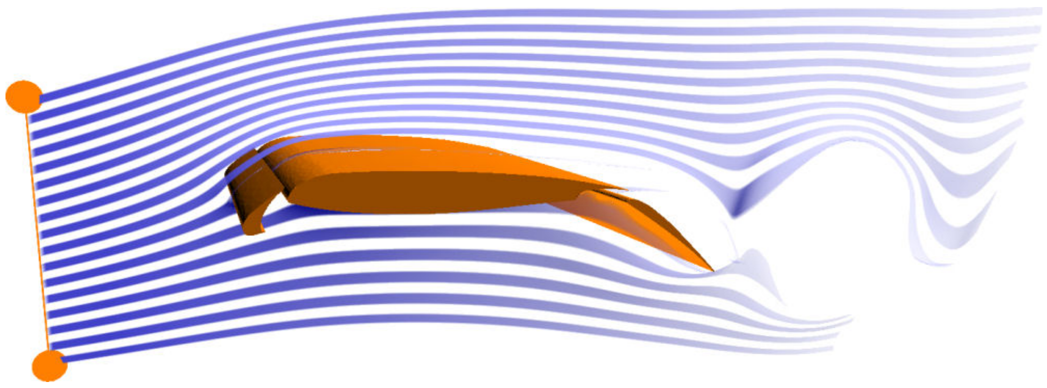


Figure 7.6: Time surface spanned by the points $\mathbf{p}_0, \ldots, \mathbf{p}_3$ and transported by the flow behind a circular cylinder. A uniform grid becomes visible by coloring some columns and rows of the internal array differently. After some integration steps, this elucidates the distortion introduced by the flow.

(a) Flow around an automobile. From [Use].



(b) Flow around a dragon fly. From [TTS+04].



(c) Flow around an airfoil visualized using smoke injected from nozzles aligned along the seeding line at the left.

Figure 7.7: Injecting smoke from nozzles is a common technique in real-world experiments to yield clearer visualizations. The upper row shows photographs from such setups. We can reproduce this with our system (lower image) either by setting alpha values to constant zero or by breaking the connectivity.

lines clearly allow to distinguish between regions of e.g. rotational and laminar behavior. Furthermore, the direction of rotation becomes visible.

7.3.3 Smoke Nozzles

In flow experiments it is common to inject smoke not from a line but from nozzles aligned in a line. Figures 7.7a-b show this. We can easily achieve this effect by setting the alpha value of every other row of the array to constant zero. This has been done in Figure 7.7c to visualize the flow around an airfoil (described in

Section 7.4.3).

However, this means that vertices are advected that will never be seen in the visualization. To avoid this, we may as well break the connectivity between adjacent rows, i.e., we do not triangulate between two rows. In fact, this splits our seeding curve into different parts which may now be placed arbitrarily in the domain.

7.3.4 Wool Tufts

In many applications it is of great interest to analyze the flow in the proximity of a boundary, e.g. where the flow might detach from the body of an airfoil or car. Such a flow separation at a boundary often indicates the presence of a recirculation zone which has a negative effect on the drag of the body. Therefore, the design goal of engineers is often to reduce flow detachment. It is commonly visualized in real-world experiments using so-called wool tufts: these are small yarns attached to the body. The different orientations and movements of wool tufts during the experiments allow the experienced viewer to draw conclusions about flow separation.

We mimic wool tufts by placing a rather high number of small seeding curves close to the boundary in a flow field. An example of this can be seen in Figure 7.10a where we did this for the flow around an airfoil (described in Section 7.4.3). In our implementation, this can easily be achieved by breaking the connectivity between adjacent rows as described earlier. The result is a set of streak ribbons visualizing the flow in the proximity of the boundary. There are two important differences to real wool tufts. First, real wool tufts have a mass whereas streak ribbons represent the movement of massless particles. Second, our streak ribbons can change their length and therefore they indicate the velocity of the flow not only in terms of direction, but also in terms of magnitude.

7.4 Results

7.4.1 Flow Behind a Circular Cylinder

Figure 7.6 and 7.8 demonstrate the results of our method applied to a flow behind a circular cylinder. The data set was derived by Bernd R. Noack (TU Berlin) from a direct numerical Navier Stokes simulation by Gerd Mutschke (FZ Rossendorf). It resolves the so called 'mode B' of the 3D cylinder wake at a Reynolds number of 300 and a spanwise wavelength of 1 diameter. The flow exhibits periodic vortex shedding leading to the well known von Kármán vortex street [ZFN$^+$95]. Figure 7.6 shows the integration of a time surface including a number of time lines. After some time, the smoke surface clearly shows the distortion introduced by the vortices in the wake of the cylinder. Figure 7.8 shows the smoke surface advected from a seeding line closely behind the cylinder. To enhance depth perception, the shadow of the smoke was projected to the back wall. Since we work with a triangular mesh, shadow computation is straightforward. Due to the periodic vortex shedding the smoke forms patterns of swirling motion after some integration steps – a clear indication of the von Kármán vortex street.

7.4.2 Flow Behind a Square Cylinder

In Figure 7.9 we visualized the flow around a confined square cylinder. This is a direct numerical Navier Stokes simulation by Simone Camarri and Maria-Vittoria Salvetti (University of Pisa), Marcelo Buffoni (Politecnico of Torino), and Angelo Iollo (University of Bordeaux I) [CSBI05] which is publicly available [Int]. It is an incompressible solution with a Reynolds number of 200 and the square cylinder has been positioned symmetrically between two parallel walls, where one of them is the wall with the shadow shown in Figure 7.9. The flow has periodic boundary conditions in spanwise direction.

In contrast to the previous cylinder data set (Section 7.4.1), this simulation is initiated from an impulsive start-up and the periodic vortex shedding develops with time. This allows us not only to study this interesting phenomenon, but also

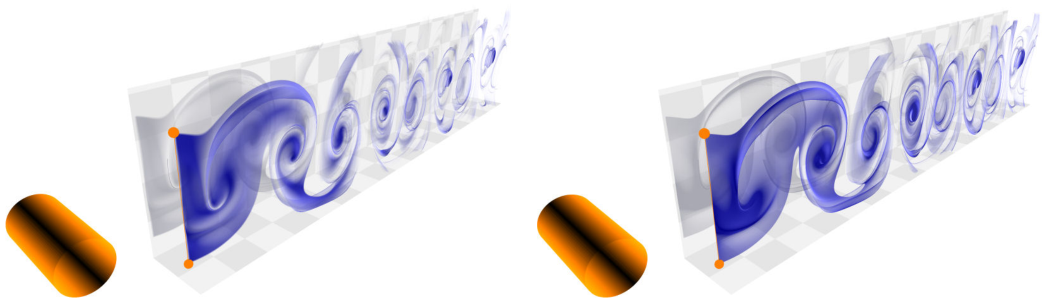

(a) 75k vertices, $\alpha_{density}$ and α_{shape}. (b) 675k vertices, $\alpha_{density}$.

Figure 7.8: Flow behind a circular cylinder. Smoke surface with different resolutions visualized with shadow to enhance depth perception.

to evaluate how our visualization technique performs with increasing unsteadiness of the flow. In order to show the alternating behavior of the vortex shedding, we seeded two smoke surfaces (25000 vertices in total) such that the red one passes above the cylinder and the blue one below.

The flow shows a rather steady behavior for the first time steps and the smoke surface develops almost like an ordinary stream surface. In fact, stream and streak surfaces coincide for steady flows. Once the vortex shedding starts, the flow becomes more unsteady – parts of the smoke are ripped off and transported downstream. Note that all the smoke in Figure 7.9 is internally represented by two smoke surfaces. This example shows that our alpha computation (Section 7.2.1) works reliable even in such challenging situations and produces convincing results. At the end of the transient, the flow develops a von Kármán vortex street with a pronounced three-dimensionality nicely captured by the smoke visualization. We conclude that the visual impression created by smoke surfaces comes closer to real smoke with increasing unsteadiness of a flow.

7.4.3 Flow Around an Airfoil

Figures 7.7c and 7.10 show the flow around a Swept-Constant-Chord-Half-model (SCCH) of an airfoil that was simulated by Bert Günther (Technical University Berlin) at a Reynolds number of 10^6 [GTP+07]. The data set exhibits periodic

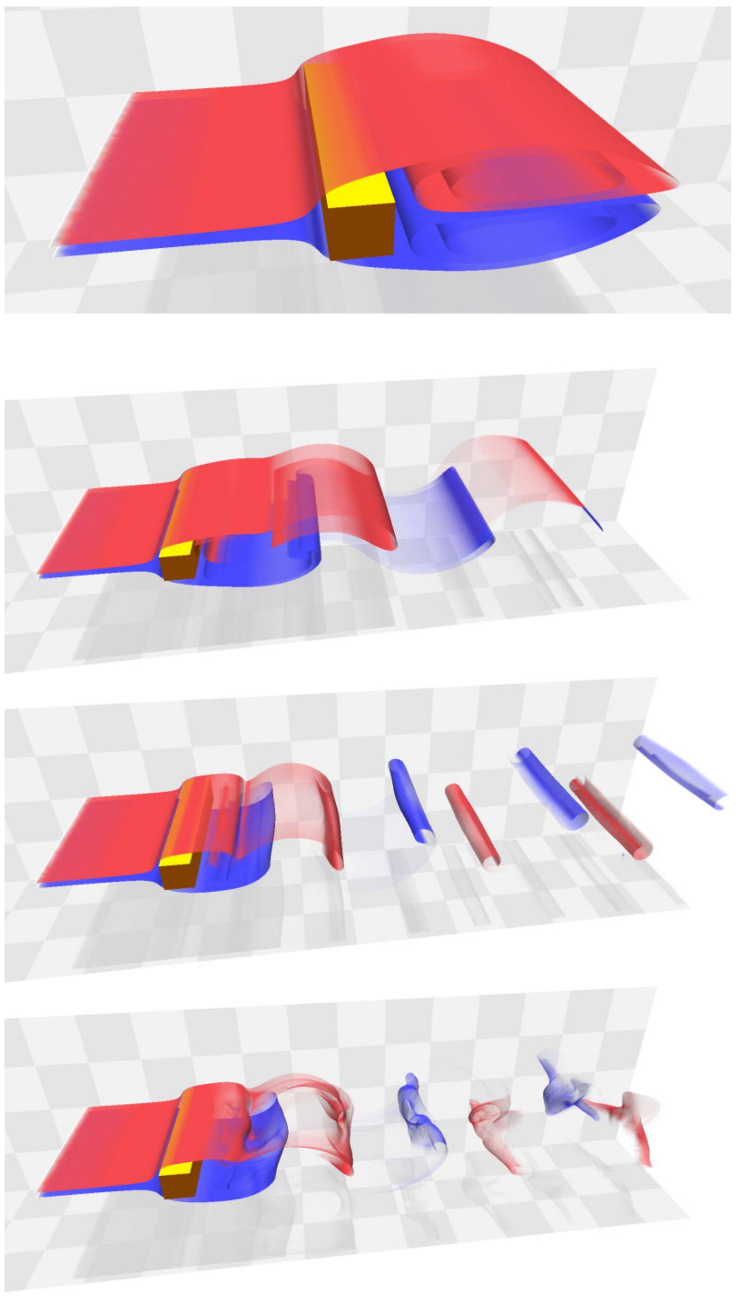


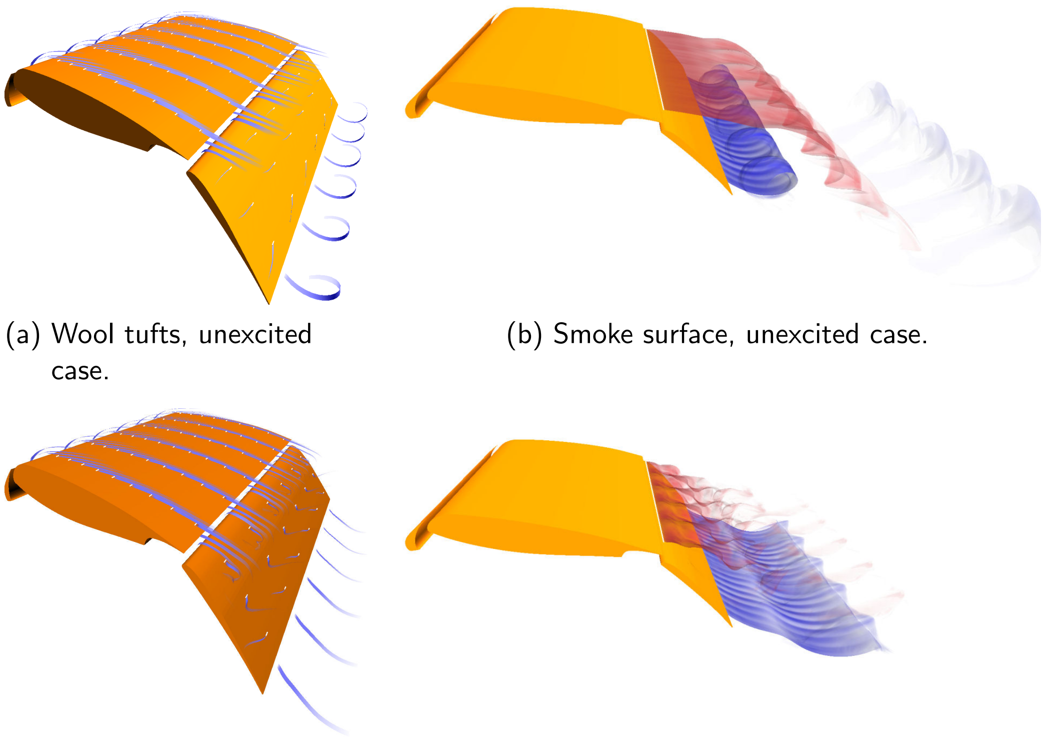
Figure 7.9: Flow behind a square cylinder. Time is increasing from top to bottom. First the smoke is gathering in a recirculation bubble behind the obstacle. After some time the shedding starts which creates vortices with alternating rotational behavior. Later, the flow develops a pronounced three-dimensionality which perfectly can be observed in the smoke structures.

boundary conditions, but note that the airfoil has a sweep angle to the incoming flow direction of 30°. The angle of attack is 6° – thereby conforming to a landing situation. The turbulence was simulated by a combined URANS and DES approach.

The wool tufts visualization of Figure 7.10a allows to describe the underlying physics of this flow. The wool tufts follow the profile on the main element of the airfoil, which means that the flow is still attached to the body. A strong jet of fluid is coming through the gap between the main element and the rear flap as shown by the wool tufts seeded under the body and reaching through the gap. The result of this jet is a detachment of the flow. This can clearly be seen from the wool tufts on the rear flap since they do not follow anymore the general flow direction. Instead, they are directed towards the viewer indicating a strong cross flow section, which is caused by the sweep angle. The wool tufts at the end of the rear flap elucidate the most prominent feature of this flow: a strong vortex created periodically at this position.

In order to increase the lift of such an airfoil, our cooperation partners from the Technical University Berlin try to avoid or minimize both the detachment of the flow at the end of the main element and the periodic vortex shedding at the rear flap. Our wool tufts visualization allows to study both phenomena and has been found useful not at least because of its interactivity. However, the wool tufts visualization shows only the flow in the proximity of the body and does not allow to study the complete characteristics of the flow. Therefore we seeded smoke surfaces close to the regions of flow detachment and vortex creation (Figure 7.10b). This allows us to study the development of these structures away from the body.

Figures 7.10c-d show the same airfoil, but now a so-called active flow control technique has been applied in order to manipulate the flow structures and achieve a higher lift. This has been done by periodically injecting air at the top of the rear flap (close to the gap). Both visualizations confirm that this excitation led to a better attached flow and less stronger vortex shedding.



(a) Wool tufts, unexcited case.

(b) Smoke surface, unexcited case.

(c) Wool tufts, excited case.

(d) Smoke surface, excited case.

Figure 7.10: Unexcited and excited flow around an airfoil. The wool tufts are mimicked using streak ribbons seeded close to the boundary of the airfoil. For example for the unexcited case, their length and orientation shows that the flow detaches from the airfoil and creates a large recirculation zone over the rear flap. Furthermore, the strong vortex created at the bottom of the rear flap becomes obvious. The smoke surfaces are rendered with 40k vertices. They clearly show that the excitation strategy is successful in diminishing the vortex created at the bottom of the rear flap.

7.4.4 Ahmed Body

Figure 7.11 shows the turbulent flow around a bluff body – the so-called Ahmed body. This data set has been computed by Erik Wassen (Technical University Berlin) using a Large-Eddy simulation scheme at a Reynolds number of 500000 based on model length and incoming velocity [WT07]. Incoming flow is assumed to come from frontal direction. The body stands on a fixed floor with a certain distance to the ground, i.e., a small layer of fluid is passing beneath the model.

The Ahmed body is a generic model for a vehicle, which has been used here in a version with a slanted rear end. The inclination angle of 25 causes a detachment of the flow and the resulting recirculation zone has a rather turbulent behavior over the ramp as it can be observed in Figure 7.11. The wool tufts visualization in Figure 7.12 clearly shows that this recirculation zone affects the vertical rear end as well: the streak ribbons point upwards from their seeding position. In contrast to this, the very last row of ribbons at the vertical rear end points in downstream direction. This indicates the presence of another shear layer created between the recirculation zone and the flow coming from under the body.

Further important structures in this flow are the two vortices created at the upper corners of the ramp. The smoke surface visualizations in Figures 7.11 and 7.13 reveal the cone-like shape of these vortices. It is known that they have a strong impact on the drag of the body. Using our smoke visualization technique we are able to infer a number of important parameters of these vortices: extent, rotation axis, as well as orientation and speed of rotation.

7.5 Evaluation

7.5.1 Performance

Since the technique is based on the integration of triangular meshes, we can achieve an interactive performance of the integration even in a CPU-based implementation. In fact, we used the capabilities of graphics hardware for the semi-transparent rendering of the triangles only. Table 7.1 shows the results of our performance measurements for smoke surfaces with different resolutions. Note that in our implementation the flow data set is given on a regular grid, making the performance depending only on the size of the streak surface and not of the data set. In fact, all visualizations in this chapter use a streak surface of 50000 or less vertices, leading to interactive frame rates in an CPU based implementation.

Vertices	Fps	Rendering	Integration	Alpha + normals
5000	85	1 ms	6 ms	4 ms
10000	44	1 ms	13 ms	9 ms
20000	25	4 ms	21 ms	15 ms
30000	18	4 ms	31 ms	21 ms
40000	14	4 ms	39 ms	29 ms
50000	11	6 ms	48 ms	37 ms

Table 7.1: Performance figures of our implementation tested on a 2.6 GHz Opteron CPU with 2 GB RAM and a GeForce 7800 GTX. *Fps* stands for frames per second, *Rendering* gives the rendering time of the semi-transparent surface, *Integration* the time for the integration of all vertices, and *Alpha + normals* the computation time for vertex normals and alpha values.

7.5.2 Correctness

In our approach we propose to represent smoke surfaces using meshes with a fixed medium resolution and fixed connectivity in order to maintain interactive frame rates. As already argued in Section 7.2, this is feasible since the physically motivated opacity $\alpha_{density}$ (7.5) gets lower with increasing triangle area, i.e., larger triangles are less visible due to the smoke metaphor anyway.

However, fixed resolution and connectivity produce artifacts e.g. where the smoke flows around obstacles or when the surface becomes strongly distorted. We addressed these problems by introducing α_{shape} and $\alpha_{curvature}$, which are designed to lower the opacity of the surface at places where it deviates too much from the "real" streak surface. Is the result still correct in the sense that all flow features like vortices are shown in the visualization?

To answer this question we produced "ground truth" visualizations using streak surfaces with excessive resolutions, rendered them using $\alpha_{density}$ only and compared the results with their medium-sized counterparts. Figure 7.8 shows this for the flow behind a circular cylinder. The resolution of the surface in Figure 7.8b is nine times larger than in Figure 7.8a. As it can be seen, both versions faithfully represent the vortices of this flow and show very similar smoke patterns.

Figure 7.13 shows such a comparison for the Ahmed data set. Although the resolution of the two surfaces differs by a factor of 16, the low-res surface nicely captures all structures of this turbulent flow. In particular, the conical vortices coming from the corners of the ramp have a disrupted smoke appearance on both sides, i.e., this is a pattern of this flow and not due to the additional opacity parameters for the low-res surface. Note that although the Ahmed body itself is symmetrical, the flow around it is not (mainly due to turbulence).

We conclude that our method produces physically correct renderings considering the given resolution when $\alpha_{density}$ is applied only. The optional parameters α_{shape} and $\alpha_{curvature}$ have been carefully designed to reduce visual clutter caused by locally too coarse resolutions while keeping the overall appearance very similar.

7.5.3 Perception

In terms of perception, the question arises: does the visualization really look like smoke? To answer this, we did not do a formal user study. However, informal reactions of a number of people confronted with the visualizations (visualization experts, flow simulation experts, and non-experts) unanimously agreed that they indeed see smoke.

Another question is whether the technique is able to detect relevant features in the flow. Here we refer to the tested applications in Section 7.4. Since part of the data sets were known in advance, we could confirm that smoke surfaces indeed were able to detect relevant features.

7.5.4 Comparison to other smoke visualization techniques

Using particle-based methods, a large number of particles would be necessary to get a detailed, surface-like smoke appearance as in Figure 7.1. Being surface-based, our method is able to get this kind of appearance directly and requires a rather low number of vertices. In contrast to particle-based representations, self-shadowing is straightforward because the surface normals are available in our mesh representation.

While volumetric methods are well-suited for the visualization of thick and diffuse smoke, thin surface-like smoke structures would require high-resolution voxel grids, which comes at the cost of memory usage and speed. In contrast to this, our surface-based approach can visualize surface-like smoke using a rather low mesh resolution.

Smoke surfaces do not intend to replace particle-based or volume-based smoke visualization techniques. However, for some situations we see advantages of smoke surfaces, making them an alternative to previous techniques. Smoke surfaces are simple: less geometric primitives are necessary to obtain expressive visualizations than for particle based or volumetric approaches. Because of this, no specialized graphics hardware is required for integration to obtain interactive frame rates. In general, smoke surfaces are the appropriate smoke representation if the smoke has an approximate surface shape, i.e., the seeding structure is a moving curve.

7.5.5 Limitations

Our method has a number of limitations. Our current implementation handles regular grids only and requests that the complete time-dependent data set is kept in main memory. However, these are not structural problems of smoke surfaces but general challenges for every interactive visualization software.

If the smoke to be visualized is known to be not surface-like but really volumetric (e.g., if the seeding structure is a volume instead of a line structure), then smoke surfaces are not the appropriate method. In these cases, particle based or volumetric approaches will give better results.

7.6 Discussion

In this chapter, we have made the following contributions:

- We introduced a new representation of smoke in a flow: as semi-transparent streak surface.

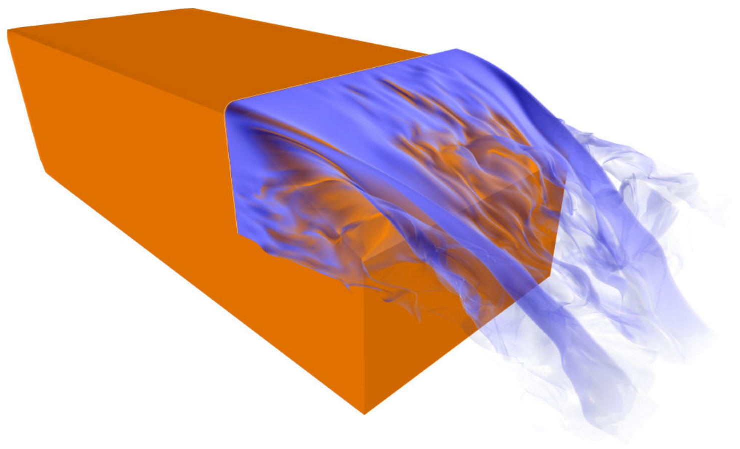


Figure 7.11: Smoke surface with 40k vertices seeded shortly before the ramp and visualized with $\alpha_{density}$, α_{shape}, $\alpha_{curvature}$, and α_{fade}.

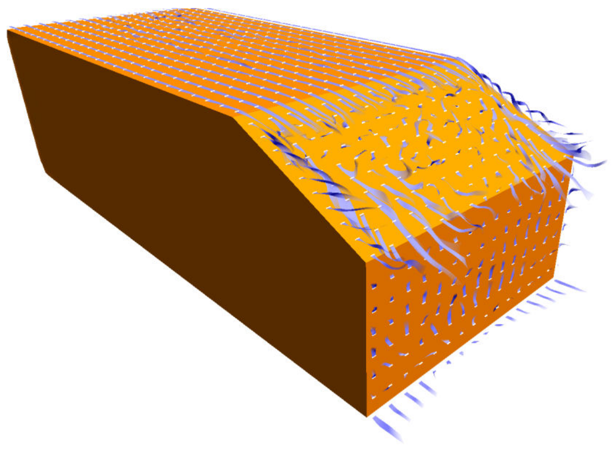


Figure 7.12: Wool tufts attached to the body. They indicate a recirculation zone at the vertical rear end where they are oriented upwards.

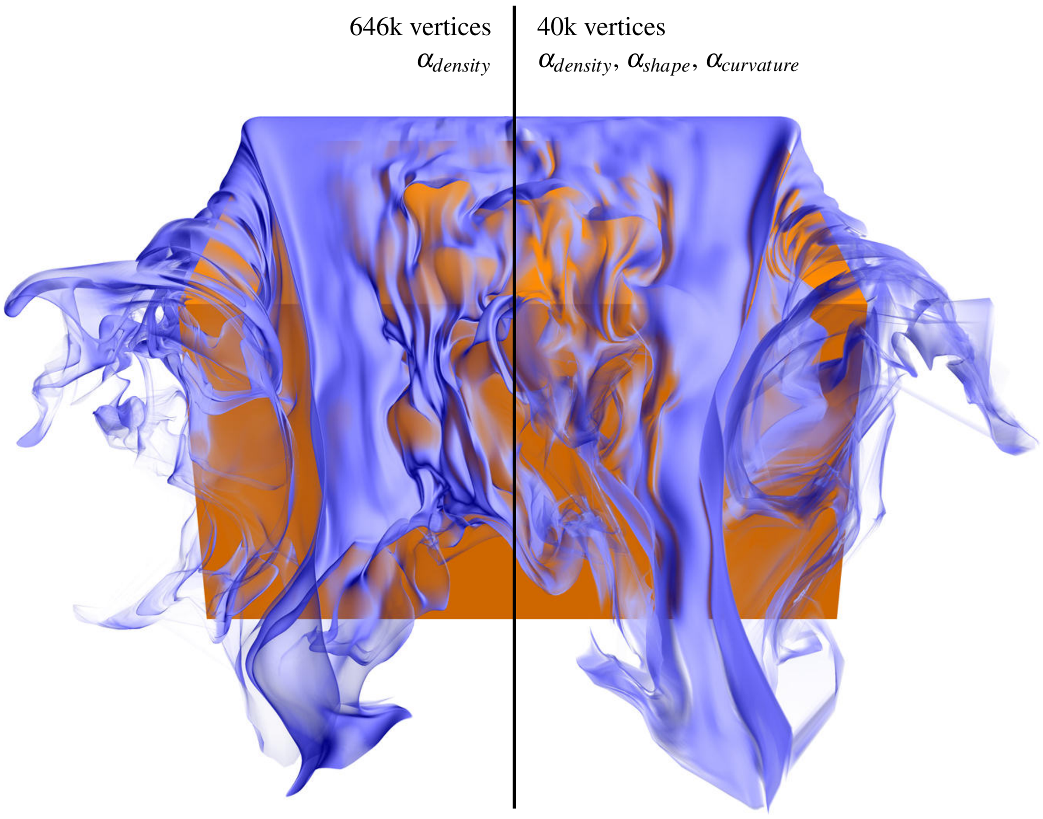


Figure 7.13: Smoke surface with different resolutions viewed from the back. The high-res surface on the left serves as a ground truth. The low-res version can be rendered interactively and shows the same smoke structures, but needs additional opacity terms to hide distorted triangles.

- For the first time, streak surfaces are used for an interactive visualization of time-dependent flow fields. This was possible by avoiding an expensive adaptive remeshing of the surface.
- By coupling the opacity of the triangle to their area, shapes, and curvatures, we obtain the impression of smoke seeded from line structures. This allows an intuitive and interactive exploration of the flow.
- By slightly changing the setup, we obtained a representation of wool tufts which are well-known from experimental flow visualization.

For future research, the performance can be further increased. Table 7.1 clearly shows that the current bottle neck is the integration and the computation time for normals and α-values. Transforming them to the GPU may further improve performance. Furthermore, the approach should be extended to handle irregular grids, and an out-of-core mechanism should be included to handle data sets which do not fit into main memory.

8 Conclusion

In this thesis, we explored several applications of vector field processing to the area of shape deformations. We investigated vector fields as a tool to deform shapes in different application scenarios. As it turned out, simple properties of the vector field often lead to useful properties in the deformation. Complex results like volume preservation can be achieved elegantly by equipping the vector field with simple mathematical properties like zero-divergence. Smooth vector fields lead to smooth deformations and time dependent flows can be used to generate complex animated smoke-like surfaces by a simple vector field integration. While we addressed many areas from the computer graphics area, like shape modeling, shape editing, simulation, animation and visualization, we believe that there is still a multitude of further applications of vector fields to shape deformations which should be addressed in future research.

Bibliography

[AB97] F. Aubert and D. Bechmann. Volume-preserving space deformation. *Comput. and Graphics*, 21(5):6125–639, 1997.

[ACWK04] 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.

[Ale03] M. Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2):105–114, 2003.

[AN05] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 87–96, 2005.

[AS07] Alexis Angelidis and Karan Singh. Kinodynamic skinning using volume-preserving deformations. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–140, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[AUGA05] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. *Recent Advances in Remeshing of Surfaces*. Springer, 2005.

[AWC04] Alexis Angelidis, Geoff Wyvill, and Marie-Paule Cani. Sweepers: Swept user-defined tools for modeling by deformation. In *Proceedings of Shape Modeling and Applications*, pages 63–73. IEEE, June 2004.

[Bar84] A. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1984. ACM Press.

[BK03a] 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.

[BK03b] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–491, 2003. (Proceedings Eurographics 2003).

[BK04] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[BK05] M. Botsch and L. Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005. (Proceedings Eurographics 2005).

[BNL06] Antoine Bouthors, Fabrice Neyret, and Sylvain Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *Eurographics Workshop on Natural Phenomena*, pages 41–50, 2006.

[Bre05] William Brennan. Smoke abstractions, 2005. http://www.pbase.com/billyb2/.

[BS04] A. Biswas and V. Shapiro. Approximate distance fields with non-vanishing gradients. *Graphical Models*, 66(3):133–159, 2004.

[BSK$^+$07] Kai Bürger, Jens Schneider, Polina Kondratieva, Jens Krüger, and Rüdiger Westermann. Interactive visual exploration of instationary 3D-flows. In *Proc. EuroVis '07*, pages 251–258, 2007.

[BWK03] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Trans. Graph.*, 22(3):862–870, 2003.

[CBC$^+$05] Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popovi¸ Physically based rigging for deformable characters. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 301–310, New York, NY, USA, 2005. ACM Press.

[CGC$^+$02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 586–593, New York, NY, USA, 2002. ACM Press.

[CHP89] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 243–252, New York, NY, USA, 1989. ACM Press.

[CK05] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):91–101, 2005.

[CLT07] Kennan Crane, Ignacio Llamas, and Sarah Tariq. Real-time simulation and rendering of 3d fluids. In *GPU Gems 3*, chapter 30, pages 633–676. Addison-Wesley Professional, 2007.

[CM93] Roger A. Crawfis and Nelson Max. Texture splats for 3D scalar and vector field visualization. In *Proc. IEEE Visualization '93*, pages 261–266, 1993.

[Col] ColDet. Free 3D collision detection library. http://photoneffect.com/coldet.

[Coq90] S. Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, New York, NY, USA, 1990. ACM Press.

[CSBI05] S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo. Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata*, 2005.

[Dav67] H. Davis. *Introduction to vector analysis*. Allyn and Bacon, Inc., Boston, 1967.

[DC96] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76. Springer-Verlag, Aug 1996. Published under the name Marie-Paule Gascuel.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, New York, NY, USA, 2001. ACM Press.

[DG95] 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.

[DSB99] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 1–8, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Eve01] Cass Everitt. Interactive-order independent transparency. *NVIDIA*, 2001. http://developer.nvidia.com/object/Interactive_Order_Transparency.html.

[Far02] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann, San Francisco, 5th edition, 2002.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[Gar85] Geoffrey Y. Gardner. Visual simulation of clouds. *SIGGRAPH Comput. Graph.*, 19(3):297–304, 1985.

[GD99] J. E. Gain and N. A. Dodgson. Adaptive refinement and decimation under free-form deformation. *Eurographics UK '99*, 1999.

[GD01] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):289–298, 2001.

[GEW05] Joachim Georgii, Florian Echtler, and Rüdiger Westermann. Interactive simulation of deformable bodies on gpus. In *Simulation and Visualisation 2005*, 2005.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: a simple framework for adaptive simulation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 281–290, New York, NY, USA, 2002. ACM Press.

[GSS99] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[GTP$^+$07] B. Günther, F. Thiele, R. Petz, W. Nitsche, J. Sahner, T. Weinkauf, and H.-C. Hege. Control of separation on the flap of a three-element high-lift configuration. In *45th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, U.S.A., January 2007. AIAA-2007-265.

[GTS$^+$04] Christoph Garth, Xavier Tricoche, Tobias Salzbrunn, Tom Bobach, and Gerik Scheuermann. Surface techniques for vortex visualization. In *VisSym*, pages 155–164, 346, 2004.

[HHK92] W. Hsu, J. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 177–184, New York, NY, USA, 1992. ACM Press.

[HML92] 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.

[Hol03] Thorsten Holtkämper. Real-time gaseous phenomena: a phenomenological approach to interactive smoke and steam. In *Proc. AFRIGRAPH '03*, pages 25–30, 2003.

[HSL$^+$06] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134, 2006.

[Hul92] J. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proc. IEEE Visualization '92*, pages 171–177, 1992.

[Int] International CFD Database, http://cfd.cineca.it/.

[JC98] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98*, pages 311–320, New York, NY, USA, 1998. ACM.

[JP99] Doug L. James and Dinesh K. Pai. Artdefo: accurate real time deformable objects. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 65–72, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[JP02] Doug L. James and Dinesh K. Pai. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 582–585, New York, NY, USA, 2002. ACM Press.

[KCVS98] 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.

[KH84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, 1984.

[KKKW05] Jens Krüger, Peter Kipfer, Polina Kondratieva, and Rüdiger Westermann. A particle system for interactive visualization of 3D flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, 2005.

[KW05] Jens Krüger and Rüdiger Westermann. GPU simulation and rendering of volumetric effects for computer games and virtual environments. *Computer Graphics Forum*, 24(3):685–693, 2005.

[LA04] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition. In *SCG '04: Proceedings of the twentieth annual sympo-*

sium on Computational geometry, pages 457–458, New York, NY, USA, 2004. ACM Press.

[LCOGL07] Yaron Lipman, Daniel Cohen-Or, Ran Gal, and David Levin. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, 26(1):5, 2007.

[LKG$^+$03] 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.

[LKM01] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 149–158, New York, NY, USA, 2001. ACM Press.

[LSCO$^+$04] 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.

[LSLCO05] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[Max95] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[MBC93] N. Max, B. Becker, and R. Crawfis. Flow volumes for interactive vector field visualization. In *Proc. Visualization 93*, pages 19–24, 1993.

[MDM$^+$02] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54, New York, NY, USA, 2002. ACM Press.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 239–246, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 471–478, New York, NY, USA, 2005. ACM Press.

[MJ96] R. MacCracken and K. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1996. ACM Press.

[MKN$^+$04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, New York, NY, USA, 2004. ACM Press.

[MW01] David Mason and Geoff 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.

[NHM97] G.M. Nielson, H. Hagen, and H. Müller. *Scientific Visualization*. IEEE Computer Society, 1997.

[NMK$^+$06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Marc Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, December 2006.

[NT98] L. P. Nedel and D. Thalmann. Real time muscle deformations using mass-spring systems. In *CGI '98: Proceedings of the Computer*

Graphics International 1998, page 156, Washington, DC, USA, 1998. IEEE Computer Society.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 291–294, New York, NY, USA, 2002. ACM Press.

[OZH00] J. O'Brien, V. Zordan, and J. Hodgins. Combining active and passive motions for secondary motion. *IEEE Computer Graphics and Applications*, 1(1):86–96, 2000.

[PB81] Stephen M. Platt and Norman I. Badler. Animating facial expressions. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 245–252, New York, NY, USA, 1981. ACM Press.

[PH06] S. Park and J. Hodgins. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics (SIGGRAPH 2006)*, 25(3), August 2006.

[PKKG03] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.

[RSB96] A. Rappoport, A. Sheffer, and M. Bercovier. Volume-preserving free-form solids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):19–27, 1996.

[SBM$^+$01] G. Scheuermann, T. Bobach, H. Hagen K. Mahrous, B. Hamann, K. Joy, and W. Kollmann. A tetrahedra-based stream surface algorithm. In *Proc. IEEE Visualization 01*, pages 151 – 158, 2001.

[SF93] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *Proc. SIGGRAPH '93*, pages 369–376, 1993.

[SF98] 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.

[SHB07] Basile Sauvage, Stefanie Hahmann, and Georges-Pierre Bonneau. Volume preservation of multiresolution meshes. *Computer Graphics Forum*, 2007. Proceedings of Eurographics'2007.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM Press.

[She02] Jonathan Richard Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Eleventh International Meshing Roundtable*, pages 115–126, 2002.

[SLCO$^+$04] 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.

[SMC04] Andrew Selle, Alex Mohr, and Stephen Chenney. Cartoon rendering of smoke animations. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 57–60, 2004.

[SP86] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160, New York, NY, USA, 1986. ACM Press.

[SP02] W. Shen and A. Pang. Tuft flow visualization. In *Proc. Second IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 705–710, 2002.

[SSEH03] Joshua Schpok, Joseph Simons, David S. Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166, 2003.

[STWE07] Tobias Schafhitzel, Eduardo Tejada, Daniel Weiskopf, and Thomas Ertl. Point-based stream surfaces and path surfaces. In *Graphics Interface*, pages 289–296, 2007.

[Tau95] G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358, New York, NY, USA, 1995. ACM Press.

[TB02] Andrzej Trembilski and Andreas Broßler. Surface-based efficient cloud visualisation for animation applications. *Journal of WSCG*, 10(2):453–460, 2002.

[TBHF03] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 68–74, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[TL07] Sarah Tariq and Ignacio Llamas. Real-time volumetric smoke, fire and water with fluid dynamics. *NVIDIA Sessions at SIGGRAPH 2007*, 2007. http://developer.nvidia.com/object/siggraph-2007.html.

[Ton98] David Love Tonnesen. *Dynamically coupled particle systems for geometric modeling, reconstruction, and animation*. PhD thesis, 1998. Adviser-Demetri Terzopoulos.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interac-*

tive techniques, pages 205–214, New York, NY, USA, 1987. ACM Press.

[TTS^{+}04] Adrian L. R. Thomas, Graham K. Taylor, Robert B. Srygley, Robert L. Nudds, and Richard J. Bomphrey. Dragonfly flight: free-flight and tethered flow visualizations reveal a diverse array of unsteady lift-generating mechanisms, controlled primarily via angle of attack. *J Exp Biol*, 207(24):4299–4323, 2004.

[TW90] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation*, 1(1):73–80, 1990.

[TW91] D. Terzopoulos and K. Waters. Modeling animated faces using scanned data. *Journal of Visualization and Computer Animation*, 2(2):123–128, 1991.

[TWHS05] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Topological methods for 2D time-dependent vector fields based on stream lines and path lines. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):383–394, 2005.

[Use] Use of Drafting in Racing, http://www.aerospaceweb.org/.

[USKS06] Tamás Umenhoffer, László Szirmay-Kalos, and Gábor Szijártó. Spherical billboards and their application to rendering explosions. In *GI '06: Proceedings of Graphics Interface 2006*, pages 57–63, 2006.

[vFTS06] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1118–1125, New York, NY, USA, 2006. ACM Press.

[vFTS07] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Explicit control of vector field based shape deformations. In *Proceedings of Pacific Graphics 2007*, 2007.

[vW92] Jarke J. van Wijk. Rendering surface-particles. In *Proc. IEEE Visualization '92*, pages 54–61, 1992.

[vW93] J. van Wijk. Implicit stream surfaces. In *Proc. Visualization 93*, pages 245–252, 1993.

[vW02] Jarke J. van Wijk. Image based flow visualization. In *Proc. ACM SIGGRAPH '02*, pages 745–754, 2002.

[Wei04] D. Weiskopf. Dye Advection Without the Blur: A Level-Set Approach for Texture-Based Visualization of Unsteady Flow. *Computer Graphics Forum (Eurographics 2004)*, 23(3):479–488, 2004.

[WT07] E. Wassen and F. Thiele. LES of wake control for a generic fastback vehicle. In *37th AIAA Fluid Dynamics Conference and Exhibit*, Miami, U.S.A, 2007. AIAA-2007-4504.

[WW92] W. Welch and A. Witkin. Variational surface modeling. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 157–166, New York, NY, USA, 1992. ACM Press.

[YZX$^+$04] 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.

[ZFN$^+$95] H.-Q. Zhang, U. Fey, B.R. Noack, M. König, and H. Eckelmann. On the transition of the cylinder wake. *Phys. Fluids*, 7(4):779–795, 1995.

[ZG05] Kok Cheong Wong Zheng Guo. Skinning with deformable chunks. *Computer Graphics Forum*, 24(3):373–382, 2005.

[ZHS$^+$05] 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.

[ZRKS05] 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.

[ZRL$^+$08] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics (ACM SIGGRAPH)*, 27(3), 2008.

[ZSH96] M. Zöckler, D. Stalling, and H.C. Hege. Interactive visualization of 3D-vector fields using illuminated stream lines. In *Proc. IEEE Visualization '96*, pages 107–113, 1996.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

Publications

Siggraph 2006

W. von Funck, H. Theisel and H.-P. Seidel: *Vector Field Based Shape Deformations*

Symposium on Geometry Processing 2007

W. von Funck, H. Theisel and H.-P. Seidel: *Elastic Secondary Deformations by Vector Field Integration*

Pacific Graphics 2007

W. von Funck, H. Theisel and H.-P. Seidel: *Explicit Control of Vector Field Based Shape Deformations*

Twelfth IMA Conference on the Mathematics of Surfaces 2007

W. von Funck, H. Theisel and H.-P. Seidel: *Implicit Boundary Control of Vector Field Based Shape Deformations*

Vision, Modeling, and Visualization 2008

W. von Funck, H. Theisel and H.-P. Seidel: *Volume-preserving Mesh Skinning*

IEEE Visualization 2008

W. von Funck, T. Weinkauf, H. Theisel and H.-P. Seidel: *Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments*