8.6 Summary and Further Reading

In this chapter we have presented several methods for repairing surface meshes, primarily distinguishing between surface-oriented and volumetric algorithms. Over the years we have observed an increasing trend towards volumetric algorithms, which by construction provide a number of guaranteed properties in the output—even if they require local resampling of the input geometry. This is why *hybrid methods* are receiving more and more attention. They combine volumetric and surface-oriented approaches in order to exploit the superior precision that is achieved from processing polygons directly but at the same time taking advantage of the robustness of volumetric voxel operations and the efficiency of hierarchical space partitions.

An important special case of the mesh repair problem that we did not address in this chapter is the removal of geometric self-intersections in 3D models that are not necessarily reflected by inconsistencies in the topological connectivity structure of the mesh. Recently, there have been some powerful algorithms proposed in response to this special case (e.g., [Campen and Kobbelt 10, Attene 10]).

As knowledge advances in the field, it is tempting to think that model repair is a necessity that will disappear if and when all other algorithms along the geometry processing pipeline would be able generate artifact-free outputs (note that ironically, some algorithms in the literature do not guarantee for their output the very properties they require of their input). However, with more and more data emerging from physical measurements and coming from heterogeneous sources, the processing of geometric data will continue to require frequent data format conversions and this will increase the probability of inconsistencies and flaws due to numerical inaccuracies. For these reasons, model repair will remain one of the most enduring problems and it will be critical for further streamlining the geometry processing pipeline.

In today's industrial design and development processes compute-intensive steps like simulation and shape analysis become faster and faster due to improved (parallel) algorithms and more powerful hardware. However, conversion and repair steps cannot be accelerated as long as the user has to control this process manually. Hence, providing automatic conversion and repair techniques will have maximum impact on the overall process optimization measured in wall clock time, which is what actually matters in industrial practice. Nonetheless, the inherent ill-posed nature of the model repair problem explains the parallel development of semiautomatic methods that provide topology control [Hétroy et al. 08] to the user.

Finally, we recommend for further reading a recent comprehensive survey by Tao Ju [Ju 09].

Deformation

In this chapter we introduce techniques for interactively deforming a given triangle mesh. This topic is challenging, since complex mathematical formulations (1) have to be hidden behind an intuitive user interface and (2) have to be implemented in a sufficiently efficient and robust manner to allow for interactive applications. This chapter gives an overview of different shape deformation techniques, classifies them into different categories, and shows their interrelations.

The deformation of a given surface $\mathcal{S}$ into the desired surface $\mathcal{S}'$ is mathematically described by a displacement function $\mathbf{d}$ that associates to each point $\mathbf{p} \in \mathcal{S}$ a displacement vector $\mathbf{d}(\mathbf{p})$. By this it maps the given surface $\mathcal{S}$ to its deformed version $\mathcal{S}'$:

$$\mathcal{S}' := \{\mathbf{p} + \mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\}.$$

For a discrete triangle mesh the displacement function $\mathbf{d}$ is piecewise linear, such that it is fully defined by the displacement vectors $\mathbf{d}_i = \mathbf{d}(\mathbf{p}_i)$ of the original mesh vertices $\mathbf{p}_i \in \mathcal{S}$.

The user controls the deformation by prescribing displacements $\bar{\mathbf{d}}_i$ for a set of so-called *handle* points $\mathbf{p}_i \in \mathcal{H} \subset \mathcal{S}$, and by constraining certain parts $\mathcal{F} \subset \mathcal{S}$ to stay *fixed* during the deformation (see Figure 9.1):

$$\begin{aligned} \mathbf{d}(\mathbf{p}_i) &= \bar{\mathbf{d}}_i, & \forall \mathbf{p}_i \in \mathcal{H}, \\ \mathbf{d}(\mathbf{p}_i) &= \mathbf{0}, & \forall \mathbf{p}_i \in \mathcal{F}. \end{aligned}$$

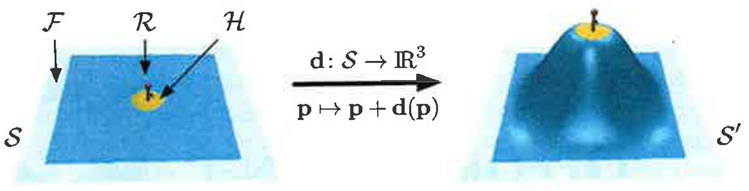

$\mathcal{F}$
$\mathcal{R}$
$\mathcal{H}$
$\mathcal{S}$
$\mathbf{d}\colon \mathcal{S} \to \mathbb{R}^3$
$\mathbf{p} \mapsto \mathbf{p} + \mathbf{d}(\mathbf{p})$
$\mathcal{S}'$

Figure 9.1. A given surface $\mathcal{S}$ is deformed into $\mathcal{S}'$ by a displacement function $\mathbf{d}(\mathbf{p})$. The user controls the deformation by moving a handle region $\mathcal{H}$ (yellow) and keeping the region $\mathcal{F}$ (gray) fixed. The unconstrained deformation region $\mathcal{R}$ (blue) should deform in an intuitive, physically-plausible manner.

The main question is how to determine the displacement vectors $\mathbf{d}_i$ for all the remaining unconstrained vertices $\mathbf{p}_i \in \mathcal{R} = \mathcal{S} \setminus (\mathcal{H} \cup \mathcal{F})$, such that the resulting shape deformation meets the user's expectations.

We discuss two classes of shape deformations in this chapter:

- We start with *surface-based deformations* in Sections 9.1–9.4. Here, the displacement function $\mathbf{d}\colon \mathcal{S} \to \mathbb{R}^3$ lives *on* the original surface $\mathcal{S}$ and is found by computations *on* the triangle mesh. These methods offer a high degree of control, since each vertex can be constrained individually. On the downside, the robustness and efficiency of the involved computations are strongly affected by the mesh complexity and the triangle quality of the original surface $\mathcal{S}$.

- *Space deformations* employ a displacement function $\mathbf{d}\colon \mathbb{R}^3 \to \mathbb{R}^3$ that warps the whole embedding space $\mathbb{R}^3$ and by this implicitly also deforms the surface $\mathcal{S}$. The deformation does not require computations on the triangle mesh $\mathcal{S}$, and therefore such methods are less affected by the complexity or triangle quality of $\mathcal{S}$. Space deformations are discussed in Sections 9.5 and 9.6.

All methods that we will describe are *linear* deformation techniques, i.e., they require the solution of a system of *linear* equations only, typically in order to minimize some quadratic deformation energy. These methods have the advantage that linear systems can be solved very efficiently (as described in the appendix). However, these methods can lead to counterintuitive results for large-scale deformations, as demonstrated in Section 9.7. Nonlinear deformation techniques overcome these limitations by minimizing more accurate nonlinear energies, which, however, requires more involved numerical schemes. We will only mention some nonlinear methods in Section 9.8.

9.1 Transformation Propagation

A simple and popular approach for shape deformation works by propagating the user-defined handle transformation within the deformation region (see Figure 9.2). After specifying the support region $\mathcal{R}$ of the deformation and a handle region $\mathcal{H}$ within it, the handle is transformed using some modeling interface. Its transformation $\mathbf{T}(\mathbf{x})$ is propagated and damped within the support region, leading to a smooth blending between the transformed handle $\mathcal{H}' = \mathbf{T}(\mathcal{H})$ and the fixed region $\mathcal{F}$.

This smooth blend is controlled by a scalar field $s\colon \mathcal{S} \to [0,1]$, which is 1 at the handle $\mathcal{H}$ (full deformation), 0 in the fixed region $\mathcal{F}$ (no deformation), and smoothly blends between 1 and 0 within the support region. One way to construct this scalar field is to compute the distances $\text{dist}_\mathcal{F}(\mathbf{p})$ and $\text{dist}_\mathcal{H}(\mathbf{p})$ from $\mathbf{p}$ to the fixed part $\mathcal{F}$ and the handle region $\mathcal{H}$, respectively, and to define

$$s(\mathbf{p}) \;=\; \frac{\text{dist}_\mathcal{F}(\mathbf{p})}{\text{dist}_\mathcal{F}(\mathbf{p}) + \text{dist}_\mathcal{H}(\mathbf{p})}. \tag{9.1}$$

The distances can be either geodesic distances on the surface [Bendels and Klein 03] or Euclidean distances in space [Pauly et al. 03], where the first typically gives better results but is more complex to compute [Kimmel and Sethian 98, Surazhsky et al. 05, Bommes and Kobbelt 07].

As an alternative, the scalar field can also be computed as a harmonic field on the surface, i.e., $\Delta s = 0$, with Dirichlet constraints 1 and 0 for the

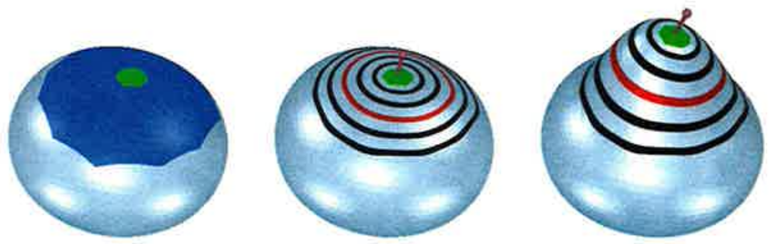


Figure 9.2. After specifying the blue support region and the green handle region (left), a smooth scalar field is constructed that is 1 at the handle and 0 outside the support (center). Its isolines are visualized in black and red, where red is the $\frac{1}{2}$-isoline. This scalar field is used to propagate and damp the handle's transformation within the support region (right). (Image taken from [Botsch et al. 06b]. ©2006 ACM, Inc. Included here by permission.)

regions $\mathcal{H}$ and $\mathcal{F}$, respectively:

$$\begin{aligned} \Delta s(\mathbf{p}_i) &= 0, \quad \mathbf{p}_i \in \mathcal{R}, \\ s(\mathbf{p}_i) &= 1, \quad \mathbf{p}_i \in \mathcal{H}, \\ s(\mathbf{p}_i) &= 0, \quad \mathbf{p}_i \in \mathcal{F}. \end{aligned} \tag{9.2}$$

Computing $s(\mathbf{p}_i)$ from the constraints in Equation (9.2) amounts to solving a linear Laplacian system (see the appendix).

While this is computationally more expensive than the distance-based scalar field of Equation (9.1), it is guaranteed to be smooth, whereas the distance-based fields can have C^1-discontinuities. The scalar field $s(\mathbf{p})$ of either (9.1) or (9.2) can be adjusted further through a transfer function $t\colon [0,1] \to [0,1]$, which provides more control and flexibility for the blending process [Pauly et al. 03].

The resulting scalar field is then used to damp the handle transformation $\mathbf{T}$ for each vertex $\mathbf{p}_i \in \mathcal{R}$ as

$$\mathbf{p}_i' \;=\; s(\mathbf{p}_i)\,\mathbf{T}(\mathbf{p}_i) \;+\; (1 - s(\mathbf{p}_i))\,\mathbf{p}_i.$$

Alternatively, the damping can be performed separately on the rotation, scale/shear, and translation components (see, e.g., [Pauly et al. 03]).

This shape deformation approach is simple and efficient to compute and yields a smooth blending of the transformed handle $\mathcal{H}$ and the fixed region $\mathcal{F}$. However, as shown in Figure 9.3, a problem of this method is that the distance-based propagation of transformations will typically not result in the geometrically most intuitive solution. This would require the smooth interpolation of the handle transformation by the displacement function $\mathbf{d}$, while otherwise minimizing some physically-motivated deformation energies, as shown in the next section.

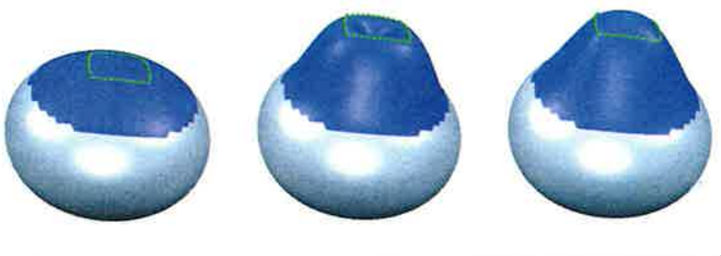


Figure 9.3. A sphere is deformed by lifting a closed handle polygon (left). Propagating this translation based on geodesic distance causes a dent in the interior of the handle polygon (center). A more intuitive solution can be achieved by minimizing physically-motivated deformation energies (right). (Image taken from [Botsch 05].)

9.2 Shell-Based Deformation

More intuitive surface deformations $\mathbf{d}$ with prescribed geometric constraints $\mathbf{d}(\mathbf{p}_i) = \bar{\mathbf{d}}_i$ can be modeled by minimizing physically-inspired deformation energies. The surface is assumed to behave like a physical skin or sheet that stretches and bends as forces are acting on it. Mathematically, this behavior can be captured by an energy functional that penalizes both stretching and bending.

Let us for the following derivations assume $\mathcal{S}$ and $\mathcal{S}'$ to be given as smooth parametric surfaces, i.e., by functions $\mathbf{p}\colon \Omega \to \mathbb{R}^3$ and $\mathbf{p}'\colon \Omega \to \mathbb{R}^3$. Similarly, the displacement function is defined as $\mathbf{d}\colon \Omega \to \mathbb{R}^3$.

As introduced in Chapter 3, the first and second fundamental forms, $\mathbf{I}(u,v)$ and $\mathbf{II}(u,v)$, can be used to measure geometrically intrinsic (i.e., parameterization independent) properties of $\mathcal{S}$, such as lengths, areas, and curvatures. When the surface $\mathcal{S}$ is deformed to $\mathcal{S}'$, such that its fundamental forms change to $\mathbf{I}'$ and $\mathbf{II}'$, the difference of the fundamental forms can be used as an elastic thin shell energy that measures stretching and bending [Terzopoulos et al. 87]:

$$\begin{aligned} E(\mathcal{S}') = \iint_\Omega & k_s \left\|\mathbf{I}'(u,v) - \mathbf{I}(u,v)\right\|_F^2 \\ & + k_b \left\|\mathbf{II}'(u,v) - \mathbf{II}(u,v)\right\|_F^2 \, \mathrm{d}u\,\mathrm{d}v. \end{aligned} \tag{9.3}$$

The stiffness parameters k_s and k_b are used to control the resistance to stretching and bending, respectively, and $\|\cdot\|_F$ is a (weighted) Frobenius norm. In a modeling scenario one has to minimize the elastic energy (9.3) subject to user-defined deformation constraints. As shown in Figure 9.1, this typically means fixing certain surface parts $\mathcal{F}$ and prescribing displacements for the handle region(s) $\mathcal{H}$.

However, minimizing the nonlinear energy (9.3) is computationally too expensive for interactive applications. It is therefore simplified by replacing the difference of fundamental forms by partial derivatives of the displacement function $\mathbf{d}$ (difference of positions) [Celniker and Gossard 91, Welch and Witkin 92]. This leads to the following *thin-shell energy*:

$$\begin{aligned} E(\mathbf{d}) = \iint_\Omega & k_s \left(\|\mathbf{d}_u(u,v)\|^2 + \|\mathbf{d}_v(u,v)\|^2\right) \\ & + \; k_b \left(\|\mathbf{d}_{uu}(u,v)\|^2 + 2\,\|\mathbf{d}_{uv}(u,v)\|^2 + \|\mathbf{d}_{vv}(u,v)\|^2\right) \mathrm{d}u\,\mathrm{d}v, \end{aligned} \tag{9.4}$$

where we use the notation $\mathbf{d}_u = \frac{\partial}{\partial u}\mathbf{d}$ and $\mathbf{d}_{uv} = \frac{\partial^2}{\partial u \partial v}\mathbf{d}$ to denote partial derivatives. Note that the stretching and bending terms of this energy are almost the same as the membrane and thin plate energies employed in

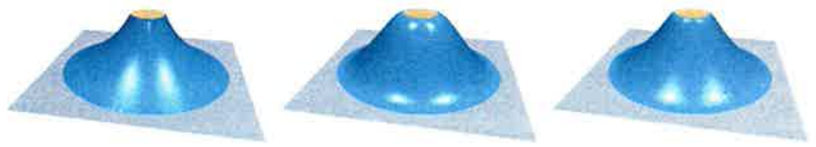


Figure 9.4. A planar surface is deformed by fixing the gray part $\mathcal{F}$, lifting the yellow handle region $\mathcal{H}$, and minimizing the shell-energy of Equation (9.4) in the blue region $\mathcal{R}$. The energy consists of stretching and bending terms, and the examples show the following: pure stretching with $k_s = 1$, $k_b = 0$, (left); pure bending with $k_s = 0$, $k_b = 1$, (center); and a weighted combination with $k_s = 1$, $k_b = 10$ (right). (Image taken from [Botsch and Sorkine 08]. ©2008 IEEE.)

Section 4.3 to measure/minimize surface area and surface curvature. The only difference is that we now use displacements $\mathbf{d}$ instead of positions $\mathbf{p}$ and by this minimize the *change of* area and *change of* curvature, i.e., we minimize stretching and bending of the surface (see Figure 9.4).

For the efficient minimization of (9.4), we apply variational calculus analogously to Section 4.3. This yields the corresponding *Euler-Lagrange* equation that characterizes the minimizer of (9.4), again subject to user constraints:

$$-k_s \Delta \mathbf{d} + k_b \Delta^2 \mathbf{d} \;=\; \mathbf{0}. \tag{9.5}$$

Hence, in order to minimize the energy (9.4), we simply have to solve the PDE (9.5). At this point we can switch back from continuous parametric surfaces to discrete triangle meshes and simply replace the continuous Laplacian in Equation (9.5) by the discrete cotangent Laplacian (3.11) introduced in Chapter 3. The bi-Laplacian can be defined recursively as the Laplacian of Laplacians:

$$\Delta^2 \mathbf{d}_i \;:=\; \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}_1(v_i)} (\cot\alpha_{i,j} + \cot\beta_{i,j})\,(\Delta \mathbf{d}_j - \Delta \mathbf{d}_i)\,,$$

where $\mathbf{d}_i = \mathbf{d}(\mathbf{p}_i) = \mathbf{d}(v_i)$ denotes the per-vertex displacements. Equation (9.5) can now be discretized to one condition per vertex:

$$\begin{array}{rcll} -k_s \Delta \mathbf{d}_i + k_b \Delta^2 \mathbf{d}_i & = & \mathbf{0}, & \mathbf{p}_i \in \mathcal{R}, \\ \mathbf{d}_i & = & \bar{\mathbf{d}}_i, & \mathbf{p}_i \in \mathcal{H}, \\ \mathbf{d}_i & = & \mathbf{0}, & \mathbf{p}_i \in \mathcal{F}. \end{array} \tag{9.6}$$

These conditions can be formulated as a linear system of equations, whose unknowns are the displacements $\mathbf{d}_1, \ldots, \mathbf{d}_n$ of the free vertices $\mathcal{R}$. The

known displacements for $\mathcal{H}$ and $\mathcal{F}$ are moved to the right-hand side $\mathbf{b}$:

$$\left[-k_s \mathbf{L} + k_b \mathbf{L}^2\right] \underbrace{\begin{pmatrix} \mathbf{d}_1^T \\ \vdots \\ \mathbf{d}_n^T \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} \mathbf{b}_1^T \\ \vdots \\ \mathbf{b}_n^T \end{pmatrix}}_{\mathbf{b}}, \tag{9.7}$$

with $\mathbf{L}$ being the Laplace matrix described in the appendix. Note that $\mathbf{x}$ and $\mathbf{b}$ are $(n \times 3)$ matrices and that the linear system therefore must be solved three times—once for each column of $\mathbf{x}$ and $\mathbf{b}$, i.e., for the x, y, and z-coordinates of the unknown displacements $\mathbf{d}_1, \ldots, \mathbf{d}_n$.

Minimizing (9.4) by solving (9.7) allows for C^1 continuous surface deformations, as can also be observed in Figure 9.4. On a discrete triangle mesh, the C^1 constraints are defined solely by the position/displacements of the first two rings of fixed vertices $\mathcal{F}$ and handle vertices $\mathcal{H}$ [Kobbelt et al. 98b]. The other vertices of $\mathcal{F} \cup \mathcal{H}$ have no influence on the solution and hence do not have to be taken into account in (9.6) and (9.7).

Interactively manipulating the handle region $\mathcal{H}$ changes the boundary constraints $\bar{\mathbf{d}}_i$ of the optimization, i.e., the right-hand side $\mathbf{B}$ of the linear system (9.7). As a consequence, this system has to be solved each time the user manipulates the handle region. In the appendix we discuss efficient linear system solvers that are particularly suited for this so-called *multiple-right-hand-side problem*. Also notice that restricting to *affine* transformation of the handle region $\mathcal{H}$ (which is usually sufficient) allows one to pre-compute basis functions of the deformation, such that instead of solving (9.7) in each frame, only the basis functions have to be evaluated [Botsch and Kobbelt 04a].

Compared to the transformation propagation of Section 9.1, the shell-based approach is computationally more expensive since a linear system must be solved for each frame. This, however, can still be performed at interactive rates using suitable linear solvers. The main advantage of the shell-based approach is that the resulting deformations are usually much more intuitive since they are derived from physical principles.

9.3 Multi-Scale Deformation

The shell-based deformation of the previous section yields physically-based, smooth and fair surface deformations. Interactive performance is achieved by simplifying the nonlinear shell energy (9.3) such that only a linear system has to be solved for the deformed surface $\mathcal{S}'$. However, as a consequence of this linearization, the method does not correctly handle fine-scale surface details, as depicted in Figure 9.5. The local rotation of geometric details

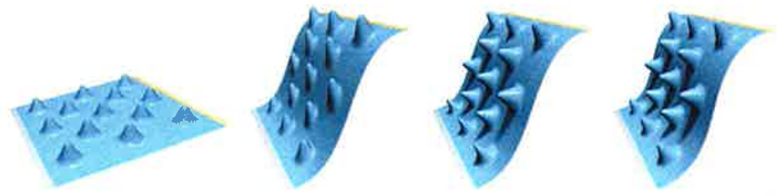


Figure 9.5. From left to right: The right strip $\mathcal{H}$ of the bumpy plane is lifted. The intuitive local rotations of geometric details cannot be achieved by a linearized deformation alone. A multi-scale approach based on normal displacements correctly rotates local details, but also distorts them, which can be seen in the leftmost row of bumps. The more accurate result of a nonlinear technique is shown. (Image taken from [Botsch and Sorkine 08]. ©2008 IEEE.)

is an inherently nonlinear behavior and therefore cannot be modeled by a purely linear technique. One way to better preserve geometric details, while still using a linear deformation approach, is to use *multi-scale* techniques, as described in this section.

The main idea of multi-scale deformations is to decompose the object into two frequency bands using the smoothing and fairing techniques introduced in Chapter 4; the low frequencies correspond to the smooth global shape, while the high frequencies correspond to the fine-scale details. Our goal is to deform the low frequencies (global shape) while preserving the high-frequency details, resulting in the desired multi-scale deformation. Figure 9.6 shows a simple 2D example of this concept.

The multi-scale deformation process is depicted in Figure 9.7. First a low-frequency representation of the given surface $\mathcal{S}$ is computed by removing the high frequencies, yielding a smooth base surface $\mathcal{B}$. The geometric details $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, i.e., the fine surface features that have been removed, represent the high frequencies of $\mathcal{S}$ and are stored as detail information.

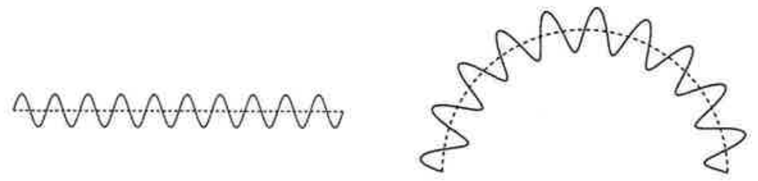


Figure 9.6. A multi-scale deformation of a sine wave. A frequency decomposition yields the dashed line as its low frequency component (left). Bending this line and adding the higher frequencies back onto it results in the desired global shape deformation (right). (Image taken from [Botsch 05].)

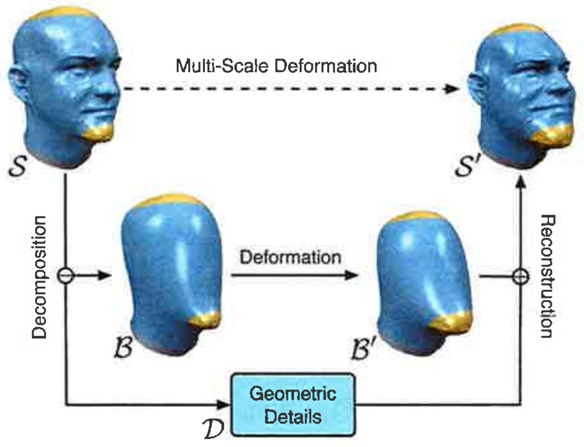


Figure 9.7. A general multi-scale editing framework consists of three main operators: the decomposition operator, which separates the low and high frequencies; the editing operator, which deforms the low frequency components; and the reconstruction operator, which adds the details back onto the modified base surface. Since the lower part of this scheme is hidden in the multi-scale kernel, only the multi-scale edit in the top row is visible to the designer. (Image taken from [Botsch and Sorkine 08]. ©2008 IEEE. Model courtesy of Cyberware.)

This allows reconstructing the original surface $\mathcal{S}$ by adding the geometric details back onto the base surface: $\mathcal{S} = \mathcal{B} \oplus \mathcal{D}$. The special operators $\ominus$ and $\oplus$ are called the *decomposition* and the *reconstruction* operator of the multi-scale framework, respectively.

This multi-scale surface representation is now enhanced by a *deformation* operator that is used to deform the smooth base surface $\mathcal{B}$ into a modified version $\mathcal{B}'$. Adding the geometric details onto the deformed base surface then results in a multi-scale deformation $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

Notice that, in general, more than one decomposition step can be used to generate a hierarchy of meshes $\mathcal{S} = \mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_k = \mathcal{B}$ with decreasing geometric complexity. In this case the frequencies that are lost from one level $\mathcal{S}_i$ to the next smoother level $\mathcal{S}_{i+1}$ are stored as geometric details $\mathcal{D}_{i+1} = \mathcal{S}_i \ominus \mathcal{S}_{i+1}$, such that after deforming the base surface to $\mathcal{B}'$, the modified original surface can be reconstructed by $\mathcal{S}' = \mathcal{B}' \oplus_{i=0}^{k-1} D_{k-i}$. Since the generalization to several hierarchy levels is straightforward, we restrict our explanations to the simpler case of a two-level decomposition, as shown in Figure 9.7.

A complete multi-scale deformation framework has to provide the three basic operators shown in Figure 9.7: the decomposition operator, the deformation operator, and the reconstruction operator. The decomposition is typically performed by mesh smoothing or fairing (Chapter 4), and for surface deformation we can employ the techniques discussed in the previous sections. The missing component is a suitable representation for the geometric detail $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, which we describe next.

9.3.1 Displacement Vectors

The straightforward representation for multi-scale details is a displacement of the base surface $\mathcal{B}$. The detail information is a vector-valued displacement function $\mathbf{h}\colon \mathcal{B} \to \mathbb{R}^3$ that associates a displacement vector $\mathbf{h}(\mathbf{b})$ to each point $\mathbf{b}$ on the base surface. In most cases $\mathcal{S}$ and $\mathcal{B}$ have the same connectivity, leading to per-vertex *displacement vectors* $\mathbf{h}_i = (\mathbf{p}_i - \mathbf{b}_i)$ [Zorin et al. 97, Kobbelt et al. 98b, Guskov et al. 99] such that

$$\mathbf{p}_i \;=\; \mathbf{b}_i + \mathbf{h}_i, \quad \mathbf{h}_i \in \mathbb{R}^3,$$

where $\mathbf{b}_i \in \mathcal{B}$ is the vertex corresponding to $\mathbf{p}_i \in \mathcal{S}$. The vectors $\mathbf{h}_i$ have to be encoded in *local frames* with regard to $\mathcal{B}$ [Forsey and Bartels 88, Forsey and Bartels 95], determined by the normal vector $\mathbf{n}_i$ and two tangent vectors $\mathbf{t}_{i,1}$ and $\mathbf{t}_{i,2}$ of the base surface $\mathcal{B}$ at point $\mathbf{b}_i$:

$$\mathbf{h}_i \;=\; \alpha_i\,\mathbf{n}_i + \beta_i\,\mathbf{t}_{i,1} + \gamma_i\,\mathbf{t}_{i,2}. \tag{9.8}$$

When the base surface $\mathcal{B}$ is deformed to $\mathcal{B}'$, the displacement vectors rotate according to the rotations of the base surface's local frames, which then leads to a plausible detail reconstruction for $\mathcal{S}'$ (see Figure 9.8):

$$\mathbf{p}'_i \;=\; \mathbf{b}'_i + \alpha_i\,\mathbf{n}'_i + \beta_i\,\mathbf{t}'_{i,1} + \gamma_i\,\mathbf{t}'_{i,2}.$$

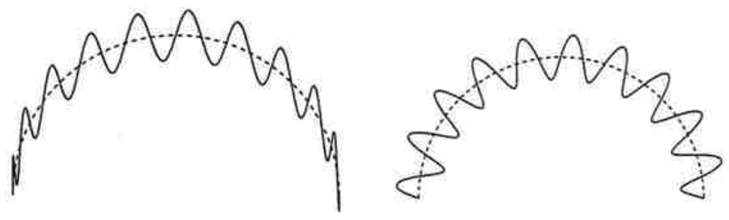


Figure 9.8. Representing the displacements with regard to the global coordinate system does not lead to the desired result (left). The geometrically intuitive solution is achieved by storing the details with regard to local frames that rotate according to the local tangent plane's rotation of $\mathcal{B}$ (right). (Image taken from [Botsch 05].)

While the normal vector $\mathbf{n}_i$ is well defined, it is not obvious how to compute the tangent axes $\mathbf{t}_{i,1}$ and $\mathbf{t}_{i,2}$. One heuristic is to choose $\mathbf{t}_{i,1}$ as the projection of the first edge incident to $\mathbf{p}_i$ into the tangent plane and to pick $\mathbf{t}_{i,2}$ to be orthogonal to $\mathbf{n}_i$ and $\mathbf{t}_{i,1}$.

9.3.2 Normal Displacements

As we will see next, long displacement vectors might lead to instabilities, in particular for bending deformations. As a consequence, the displacement vectors should be as short as possible, which is the case if they connect vertices $\mathbf{p}_i \in \mathcal{S}$ to their *closest* surface points on $\mathcal{B}$ instead of to their corresponding vertices $\mathbf{b}_i \in \mathcal{B}$.

This idea leads to *normal displacements* that are perpendicular to $\mathcal{B}$, i.e., parallel to its normal field $\mathbf{n}(\mathbf{b})$:

$$\mathbf{p}_i \;=\; \mathbf{b}_i + h_i \cdot \mathbf{n}_i, \quad h_i \in \mathbb{R}.$$

Since the per-vertex displacements $\mathbf{h}_i$ of (9.8) are in general not parallel to the surface normal $\mathbf{n}_i$, normal displacements require a re-sampling of either $\mathcal{S}$ or $\mathcal{B}$. Shooting rays in normal direction from each base vertex $\mathbf{b}_i \in \mathcal{B}$ and deriving new vertex positions $\mathbf{p}_i$ as their intersections with the detailed surface $\mathcal{S}$ leads to a resampling of the latter [Guskov et al. 00, Lee et al. 00]. Because $\mathcal{S}$ might be a detailed surface with high-frequency features, such a resampling is likely to introduce alias artifacts.

Hence we go the other direction, following [Kobbelt et al. 99b]. For each point $\mathbf{p}_i \in \mathcal{S}$, a local Newton iteration finds a base point $\mathbf{b}_i \in \mathcal{B}$ such that $(\mathbf{p}_i - \mathbf{b}_i)$ is parallel to the surface normal $\mathbf{n}_i = \mathbf{n}(\mathbf{b}_i)$. Note that $\mathbf{b}_i$ is now an arbitrary surface point on $\mathcal{B}$ (not necessarily a vertex). This point is contained in a triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \subset \mathcal{B}$ and therefore can be represented by barycentric interpolation (see Equation (1.3)):

$$\mathbf{b}_i \;=\; \alpha\,\mathbf{a} + \beta\,\mathbf{b} + \gamma\,\mathbf{c}.$$

Its normal vector $\mathbf{n}_i$ is computed by barycentric interpolation of the vertex normals, similar in concept to Phong shading :

$$\mathbf{n}_i \;=\; \frac{\alpha\,\mathbf{n_a} + \beta\,\mathbf{n_b} + \gamma\,\mathbf{n_c}}{\|\alpha\,\mathbf{n_a} + \beta\,\mathbf{n_b} + \gamma\,\mathbf{n_c}\|}.$$

Using this continuous normal field $\mathbf{n}(\mathbf{b})$ on the base surface $\mathcal{B}$, a local Newton iteration can find the barycentric coordinates (α, β, γ) of the base point $\mathbf{b}_i$ as the root of the function

$$f(\alpha, \beta, \gamma) \;=\; (\mathbf{p}_i - \mathbf{b}_i) \times \mathbf{n}_i.$$

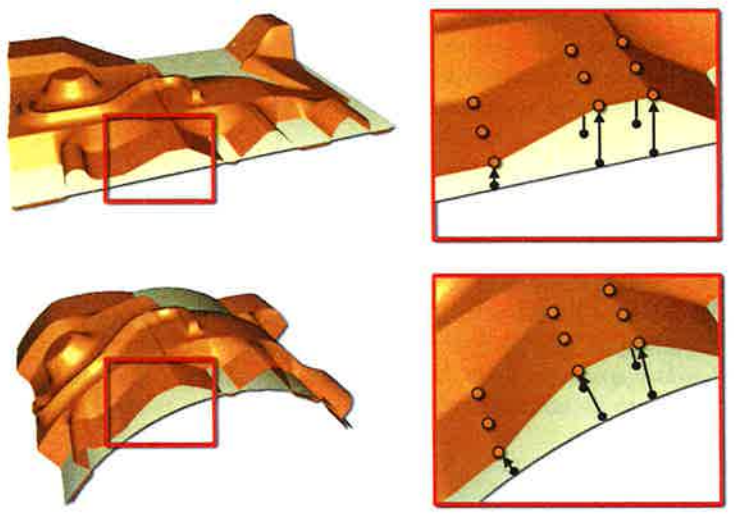


Figure 9.9. Top: The original surface $\mathcal{S}$ (orange) is decomposed into a low-frequency base surface $\mathcal{B}$ (yellow) and high-frequency normal displacements $\mathcal{D}$ (top right). Bottom: When the base surface is deformed to $\mathcal{B}'$, the normal vectors rotate accordingly and the displaced surface $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$ gives the desired result.

The process is initialized with the triangle closest to $\mathbf{p}_i$. If a barycentric coordinate becomes negative during the Newton iteration, one proceeds to the respective neighboring triangle.

Once the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ and the barycentric coordinates (α, β, γ) have been found, the deformed point $\mathbf{p}_i'$ can be efficiently computed as a normal displacement of the deformed base surface $\mathcal{B}'$ (see Figure 9.9):

$$\mathbf{p}_i' \;=\; \left(\alpha\,\mathbf{a}' + \beta\,\mathbf{b}' + \gamma\,\mathbf{c}'\right) \;+\; h_i \cdot \frac{\alpha\,\mathbf{n}_\mathbf{a}' + \beta\,\mathbf{n}_\mathbf{b}' + \gamma\,\mathbf{n}_\mathbf{c}'}{\|\alpha\,\mathbf{n}_\mathbf{a}' + \beta\,\mathbf{n}_\mathbf{b}' + \gamma\,\mathbf{n}_\mathbf{c}'\|}.$$

This process avoids a resampling of $\mathcal{S}$ and therefore allows for the preservation of all of its sharp features (see also [Pauly et al. 06] for a comparison and discussion). Since the base points $\mathbf{b}_i$ are arbitrary surface points of $\mathcal{B}$, the connectivity of $\mathcal{S}$ and $\mathcal{B}$ is no longer restricted to be identical. This can be exploited in order to remesh the base surface $\mathcal{B}$ for the sake of higher numerical robustness [Botsch and Kobbelt 04b].

The difference in length of general displacement vectors and normal displacements typically depends on how much $\mathcal{B}$ differs from $\mathcal{S}$. For instance, in Figure 9.7 the general displacements are in average about 9 times longer than normal displacements. Besides being shorter, normal displacements

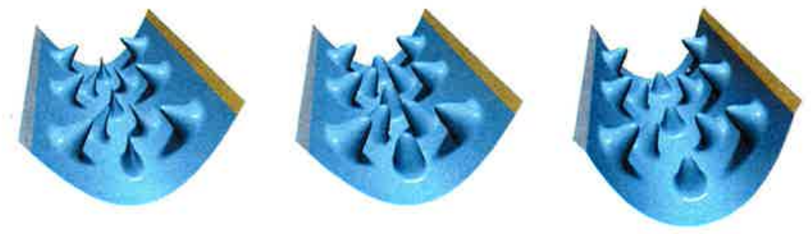


Figure 9.10. For the bending of the bumpy plane, normal displacements distort geometric details and almost lead to self-intersections (left), whereas displacement volumes (center) and deformation transfer (right) achieve more natural results. (Image taken from [Botsch et al. 06c].)

have the additional advantage that they do not require the heuristic computation of the tangent directions $\mathbf{t}_{i,1}$ and $\mathbf{t}_{i,2}$.

9.3.3 Advanced Techniques

While normal displacements are extremely efficient, their main problem is that neighboring displacement vectors are not coupled in any way. When strongly bending the surface in a convex or concave manner, the angle between neighboring displacement vectors increases or decreases, leading to an undesired distortion of geometric details (see Figures 9.5 and 9.10). In the extreme case of neighboring displacement vectors crossing each other (which happens if the curvature of $\mathcal{B}'$ becomes larger than the displacement length h_i), the surface even self-intersects locally. These problems are addressed by the advanced detail encoding techniques sketched in this section.

Displacement volumes. Both problems—the unnatural change of the volume and the local self-intersections—are addressed by displacement *volumes* instead of displacement *vectors* [Botsch and Kobbelt 03]. Each triangle $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$ of $\mathcal{S}$, together with the corresponding points $(\mathbf{b}_i, \mathbf{b}_j, \mathbf{b}_k)$ on $\mathcal{B}$, defines a triangular prism. The volumes of those prisms are used as detail coefficients $\mathcal{D}$ and are kept constant during deformations. For a modified base surface $\mathcal{B}'$, the reconstruction operator therefore has to find $\mathcal{S}'$ such that the enclosed prisms have the same volumes as the original shape. The local volume preservation leads to more intuitive results and avoids local self-intersections (see Figures 9.5 and 9.10). However, the improved detail preservation comes at the price of higher computational cost compared to a linear detail reconstruction process.

Deformation transfer. [Botsch et al. 06c] use the deformation transfer approach of [Sumner and Popović 04] to transfer the base surface deformation $\mathcal{B} \mapsto \mathcal{B}'$ onto the detailed surface $\mathcal{S}$, resulting in a multi-scale deformation $\mathcal{S}'$. This method yields results similar in quality to displacement volumes (see Figures 9.5 and 9.10), but it only requires solving a sparse linear Poisson system. Both in terms of results and of computational efficiency, this method can be considered as lying in between displacement vectors and displacement volumes.

9.4 Differential Coordinates

While multi-scale deformation is an effective tool for enhancing shape deformations by fine-scale detail preservation, the generation of such a hierarchy can become quite involved for geometrically or topologically complex models. To avoid the explicit multi-scale decomposition, another class of methods modifies differential surface properties instead of spatial coordinates and then reconstructs a deformed surface having the desired differential coordinates. We first describe two typical differential representations—gradients and Laplacians—and how to derive the deformed surface from the manipulated differential coordinates. We then explain how to compute the local transformations of differential coordinates based on the user's deformation constraints.

9.4.1 Gradient-Based Deformation

Gradient-based methods [Yu et al. 04, Zayer et al. 05a] deform the surface by manipulating the original surface gradients and then finding the deformed surface that matches the target gradient field in the least-squares sense. This two-step deformation process is depicted by Figure 9.11.

For the manipulation of gradients, let us first consider a piecewise linear function $f\colon \mathcal{S} \to \mathbb{R}$ that lives on the original mesh and is defined by its values f_i at the mesh vertices. Its gradient $\nabla f\colon \mathcal{S} \to \mathbb{R}^3$ is a piecewise constant vector field, i.e., a constant vector $\mathbf{g}_T \in \mathbb{R}^3$ for each triangle T.

If instead of a scalar function f the piecewise linear coordinate function $\mathbf{p}\colon \mathcal{S} \to \mathbb{R}^3$, $v_i \mapsto \mathbf{p}_i$, is considered, then the gradient within a face T is a constant 3×3 Jacobian matrix:

$$\nabla \mathbf{p}|_T = \begin{bmatrix} \nabla \mathbf{p}_x|_T \\ \nabla \mathbf{p}_y|_T \\ \nabla \mathbf{p}_z|_T \end{bmatrix} =: \mathbf{J}_T \in \mathbb{R}^{3\times 3}.$$

The rows of $\mathbf{J}_T$ are just the gradients of the x-, y-, and z-coordinates of

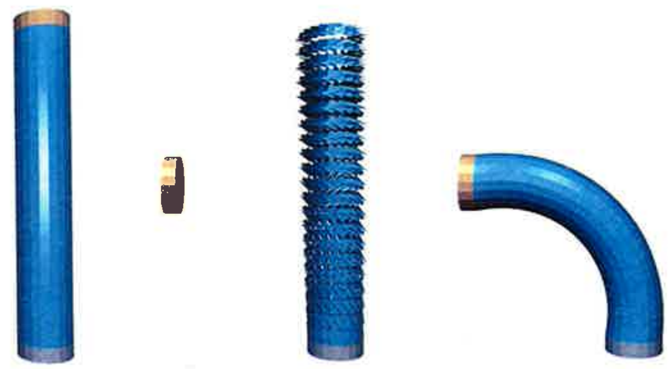


Figure 9.11. Using gradient-based editing to bend the cylinder by 90° (left). Rotating the handle and propagating its damped local rotation to the individual triangles (resp. their gradients $\mathbf{J}_T$) breaks up the mesh (center), but solving the Poisson system (9.10) reconnects it and yields the desired result (right). (Image taken from [Botsch and Sorkine 08]. ©2008 IEEE.)

the function $\mathbf{p}$ within triangle T, respectively, which can be computed by Equation (3.9).

The face gradients $\mathbf{J}_T$ are then modified by multiplying them by a 3×3 matrix $\mathbf{M}_T$ that represents the desired local rotation/scale/shear for the triangle T, yielding the new, desired gradients $\mathbf{J}_T'$:

$$\mathbf{J}_T' = \mathbf{M}_T \, \mathbf{J}_T.$$

How to actually determine the local transformation $\mathbf{M}_T$ from the user-defined handle transformation is discussed in Section 9.4.3. For a better understanding, Figure 9.11 (center) shows the transformations $\mathbf{M}_T$ applied to the individual triangles T, thereby breaking up the mesh.

The remaining step is to find new vertex positions $\mathbf{p}_i'$, such that the gradients $\nabla \mathbf{p}'|_T$ of the deformed mesh are as close as possible to the target gradients $\mathbf{J}_T'$. Intuitively this means reconnecting the triangles of Figure 9.11 (center) while changing their orientations as little as possible.

In the continuous setting, the analogous problem would be to find a function $f\colon \Omega \to \mathbb{R}$ that best matches a given gradient field $\mathbf{g}$. This amounts to minimizing the following energy functional:

$$E(f) = \iint_\Omega \|\nabla f(u,v) - \mathbf{g}(u,v)\|^2 \, du\,dv.$$

Applying variational calculus yields the Euler-Lagrange equation

$$\Delta f = \operatorname{div} \mathbf{g}, \tag{9.9}$$

which has to be solved for the optimal function f. Replacing f by the x-, y-, and z-coordinates of the deformed vertices $\mathbf{p}'_i$ and discretizing (9.9) by the discrete Laplace (3.11) and divergence (3.12) yields the linear system

$$\mathbf{L} \cdot \begin{pmatrix} \mathbf{p}_1'^T \\ \vdots \\ \mathbf{p}_n'^T \end{pmatrix} = \begin{pmatrix} \operatorname{div} \mathbf{J}'(v_1) \\ \vdots \\ \operatorname{div} \mathbf{J}'(v_n) \end{pmatrix}. \tag{9.10}$$

This linear system is solved three times for the x-, y-, and z-coordinates of the deformed vertices, with the right hand side being the divergence of the modified x-, y-, and z-gradients (the rows of the modified Jacobians $\mathbf{J}'$). Note that, analogously to Equation (9.7), proper constraints have to be employed to make the system non-singular, e.g., by prescribing positions $\mathbf{p}'_i$ for handle and fixed vertices in $\mathcal{H}$ or $\mathcal{F}$.

Comparing Equation (9.10) to Equation (9.7), gradient-based editing must solve a Poisson system only, which is sparser and hence slightly more efficient than solving the bi-Laplacian system of the shell-based approach. On the other hand, the Poisson system allows for C^0 continuity at the boundary of the deformed region $\mathcal{R}$ only, whereas the shell-based approach yields C^1 continuous deformations.

9.4.2 Laplacian-Based Deformation

The second class of deformation methods based on differential coordinates is *Laplacian editing* [Lipman et al. 04, Sorkine et al. 04, Zhou et al. 05, Nealen et al. 05]. The setting is very similar to the gradient-based editing of the previous section, but now we manipulate per-vertex Laplacians instead of per-face gradients. We first compute initial Laplace coordinates $\boldsymbol{\delta}_i = \Delta(\mathbf{p}_i)$, manipulate them to $\boldsymbol{\delta}'_i = \mathbf{M}_i\,\boldsymbol{\delta}_i$ as discussed in Section 9.4.3, and find new coordinates $\mathbf{p}'_i$ that match the target Laplacian coordinates.

In the continuous setting this problem amounts to minimizing

$$E(\mathbf{p}') = \iint_\Omega \left\| \Delta\mathbf{p}'(u,v) - \boldsymbol{\delta}'(u,v) \right\|^2 \,\mathrm{d}u\,\mathrm{d}v,$$

which leads to the Euler-Lagrange equation

$$\Delta^2\,\mathbf{p}' = \Delta\boldsymbol{\delta}'.$$

For a discrete triangle mesh, this yields a bi-Laplacian system that has to be solved for the x-, y-, and z-coordinates of the deformed vertices $\mathbf{p}'_i$:

$$\mathbf{L}^2 \cdot \begin{pmatrix} \mathbf{p}_1'^T \\ \vdots \\ \mathbf{p}_n'^T \end{pmatrix} = \begin{pmatrix} \Delta\boldsymbol{\delta}_1'^T \\ \vdots \\ \Delta\boldsymbol{\delta}_n'^T \end{pmatrix}.$$

Again, suitable boundary constraints for $\mathcal{F}$ and $\mathcal{H}$ must be employed. Note that although the original works use the uniform Laplacian (3.10), the cotangent Laplacian (3.11) has been shown to yield better results for irregular meshes [Botsch and Sorkine 08].

There is an interesting connection of Laplacian-based deformation to the shell-based deformation of Section 9.2. Let us neglect for a moment the local transformations $\boldsymbol{\delta}_i \mapsto \boldsymbol{\delta}'_i$ and instead compute the new coordinates $\mathbf{p}'_i$ from the *original* Laplacians $\boldsymbol{\delta}_i$, i.e., by solving $\Delta^2\mathbf{p}' = \Delta\boldsymbol{\delta}$, again imposing constraints $\mathbf{p}'_i = \bar{\mathbf{p}}_i$ for $\mathbf{p}_i \in \mathcal{H} \cup \mathcal{F}$. Using the two identities $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ and $\boldsymbol{\delta} = \Delta\mathbf{p}$ reveals that the latter PDE is equivalent to the Euler-Lagrange equation $\Delta^2\mathbf{d} = \mathbf{0}$ of the shell-based approach. As a consequence, the two methods are equivalent up to the way they model the local rotations of geometric details or differential coordinates, respectively, employing either a multi-scale technique (Section 9.3) or local transformations of Laplacians, as discussed next. Another consequence is that Laplacian editing yields C^1 continuous deformation, in contrast to the C^0 deformations of gradient-based editing.

9.4.3 Local Transformations

The missing component for gradient-based or Laplacian-based deformation is a technique for modifying the gradients $\mathbf{J}_T$ or Laplacians $\boldsymbol{\delta}_i$ based on the handle transformation provided by the user. The methods discussed below derive local per-vertex or per-face transformations $\mathbf{M}_i$ or $\mathbf{M}_T$, respectively, in order to transform gradients or Laplacians, as discussed above:

$$\mathbf{J}'_T = \mathbf{M}_T\,\mathbf{J}_T \qquad \text{or} \qquad \boldsymbol{\delta}'_i = \mathbf{M}_i\,\boldsymbol{\delta}_i.$$

Propagation of deformation gradients. The first approach is to transform the differential coordinates by the gradient of the handle transformation, which is interpolated over the deformable region similar to Section 9.1 [Yu et al. 04, Zayer et al. 05a]. Typically, the user manipulates the handle by prescribing an affine transformation

$$\mathbf{T}(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{t}.$$

The gradient of $\mathbf{T}(\mathbf{x})$ is the constant 3×3 matrix $\mathbf{M}$, which represents the rotation and scale/shear components of the handle transformation.

We would like to propagate this matrix over the deformable region and damp it using the smooth scalar field $s\colon \mathcal{S} \to [0,1]$ from Section 9.1 such that we smoothly blend from the full transformation $\mathbf{M}$ at the handle $\mathcal{H}$ to no transformation $\mathbf{Id}$ at the fixed region $\mathcal{F}$.

However, since rotations should be interpolated differently than scalings, these two components have to be separated first. The tool to decompose the matrix $\mathbf{M}$ into rotation $\mathbf{R}$ and scale/shear $\mathbf{S}$ is the so-called *polar* *decomposition* [Shoemake and Duff 92]. After computing the singular value decomposition $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we can find rotation and scale/shear as

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \qquad \text{and} \qquad \mathbf{S} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T.$$

Since $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices we get

$$\mathbf{R}\mathbf{S} = \mathbf{U}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{M}.$$

The rotation and scaling components are then interpolated separately over the deformable region, yielding the damped local transformation $\mathbf{M}_i$ at vertex v_i

$$\mathbf{M}_i = \text{slerp}(\mathbf{R}, \mathbf{Id}, s_i) \cdot ((1 - s_i)\,\mathbf{S} + s_i\,\mathbf{Id}),$$

where slerp$(\cdot)$ denotes quaternion interpolation between the full rotation $\mathbf{R}$ and the identity matrix $\mathbf{Id}$, and $s_i = s(\mathbf{p}_i)$ is the vertex blending value. The local transformation $\mathbf{M}_T$ for a face $T = (\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$ is computed using the blending value $s_T = (s_i + s_j + s_k)/3$.

By construction this method works very well for rotations (see Figure 9.11), but it unfortunately is insensitive to handle translations. Adding a translation $\mathbf{t}$ to a given handle deformation $\mathbf{T}(\mathbf{x})$ does *not* change its gradient $\mathbf{M}$ and thus has no influence on the resulting surface gradients $\mathbf{J}'_T$ or Laplacian coordinates $\boldsymbol{\delta}'_i$. But because there is a (nonlinear) connection between handle translations and local rotations of the surface, these methods will give counter-intuitive results for deformations containing large translations (see also Section 9.7).

Implicit optimization. Sorkine and colleagues simultaneously optimize for both the new vertex positions $\mathbf{p}'_i$ and the local rotations $\mathbf{M}_i$ by minimizing the energy functional [Sorkine et al. 04]

$$E(\mathbf{p}') = \sum_{i=1}^{n} A_i \,\|\mathbf{M}_i\,\boldsymbol{\delta}_i - \Delta\mathbf{p}'_i\|^2, \tag{9.11}$$

where $A_i = A(v_i)$ is the local vertex area. In this equation the transformations $\mathbf{M}_i = \mathbf{M}_i(\mathbf{p}')$ depend on the new vertex positions $\mathbf{p}'_j$. Note that boundary constraints for $\mathcal{H}$ and $\mathcal{F}$ again have to be prescribed.

To avoid a nonlinear optimization, which would be necessary for *rigid transformations* $\mathbf{M}_i$ (i.e., rotations), the local transformations are restricted to *linearized similarity transformations*. These can be represented by skew-symmetric matrices

$$\mathbf{M}_i = \begin{bmatrix} s_i & -h_{i,z} & h_{i,y} \\ h_{i,z} & s_i & -h_{i,x} \\ -h_{i,y} & h_{i,x} & s_i \end{bmatrix}.$$

The parameters $(s_i, \mathbf{h}_i)$ can be determined by writing down the desired transformation constraints

$$\mathbf{M}_i(\mathbf{p}_i - \mathbf{p}_j) = \mathbf{p}'_i - \mathbf{p}'_j, \quad \forall \mathbf{p}_j \in \mathcal{N}_1(\mathbf{p}_i)$$

and extracting $(s_i, \mathbf{h}_i)$ as linear combinations of $\mathbf{p}'_i$. The precise derivation can be found in [Sorkine et al. 04]. Plugging the linear expression for $\mathbf{M}_i$ back into (9.11) leads to a linear least-squares problem, which can be solved efficiently. On the downside, however, the linearized transformations lead to artifacts in the case of large rotations (see Section 9.7).

9.5 Freeform Deformation

All deformation approaches described so far are *surface-based*: they compute a smooth deformation field *on* the surface $\mathcal{S}$ by minimizing some quadratic energy, which amounts to solving a linear system corresponding to the respective Euler-Lagrange equation.

An apparent drawback of such methods is that their computational effort and numerical robustness are strongly related to the complexity and quality of the surface tessellation. In the presence of degenerate triangles, the discrete cotangent weights (3.11) for the Laplacian operator are not well defined and thus the involved linear systems become singular. Similarly, topological artifacts like gaps or non-manifold configurations lead to problems, since local vertex neighborhoods become inconsistent. In such cases quite some effort has to be spent to still be able to compute smooth deformations, like eliminating degenerate triangles (Chapter 8) or even remeshing the complete surface (Chapter 6). But even if the mesh quality is sufficiently high, extremely complex meshes will result in linear systems that cannot be solved due to their sheer size.

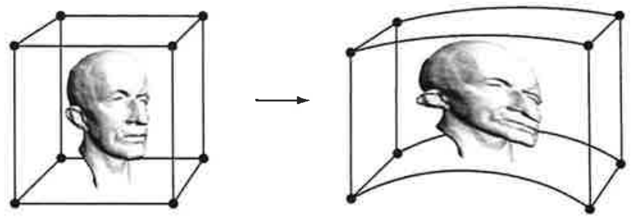


Figure 9.12. Space deformations warp the embedding space around an object and thus implicitly deform the object. (Image taken from [Botsch et al. 06b]. ©2006 ACM, Inc. Included here by permission.)

These problems are avoided by *space deformations*, which deform the ambient space and thus implicitly deform the embedded objects (see Figure 9.12). In contrast to surface-based methods, space deformation approaches employ a trivariate deformation function $\mathbf{d}\colon \mathbb{R}^3 \to \mathbb{R}^3$ to transform all points of the original surface $\mathcal{S}$. Since the space deformation function $\mathbf{d}$ does not depend on a particular surface representation, it can be used to deform all kinds of explicit surface representations, e.g., by transforming all vertices of a triangle mesh or all points of a point-sampled model.

9.5.1 Lattice-Based Freeform Deformation

Classical freeform deformation (FFD) [Sederberg and Parry 86] represents the space deformation by a trivariate tensor-product spline function

$$\mathbf{d}(u,v,w) \;=\; \sum_i \sum_j \sum_k \boldsymbol{\delta}\mathbf{c}_{ijk}\, N_i(u)\, N_j(v)\, N_k(w)\,, \qquad (9.12)$$

where N_i are B-spline basis functions and $\boldsymbol{\delta}\mathbf{c}_{ijk} = (\mathbf{c}'_{ijk} - \mathbf{c}_{ijk})$ are the displacements of the control points $\mathbf{c}_{ijk}$ (compare this to tensor-product spline surfaces of Section 1.3.1). Let us for the sake of simpler notation order the grid points and basis functions in a linear manner:

$$\boldsymbol{\delta}\mathbf{c}_l := \boldsymbol{\delta}\mathbf{c}_{ijk} \quad \text{and} \quad N_l(\mathbf{u}) = N_l(u,v,w) := N_i(u)\, N_j(v)\, N_k(w)\,.$$

This allows us to re-write Equation (9.12) as

$$\mathbf{d}(\mathbf{u}) \;=\; \sum_{l=1}^{n} \boldsymbol{\delta}\mathbf{c}_l\, N_l(\mathbf{u})\,.$$

Each original vertex $\mathbf{p}_i \in \mathcal{S}$ has a corresponding parameter value $\mathbf{u}_i = (u_i, v_i, w_i)$ such that $\mathbf{p}_i = \sum_l \mathbf{c}_l N_l(\mathbf{u}_i)$. The vertex is then transformed by $\mathbf{p}'_i = \mathbf{p}_i + \mathbf{d}(\mathbf{u}_i)$, which can be computed efficiently since $N_l(\mathbf{u}_i)$ stays constant and can be precomputed.

The deformation can be controlled by manipulating the positions of control points, i.e., by prescribing the control point displacements $\boldsymbol{\delta}\mathbf{c}_l$ (see Figure 9.13 (left)). This, however, can become tedious for more complex control grids. Moreover, the support of the deformation is sometimes difficult to predict since it is determined as the intersection of a volumetric basis function's support with the surface $\mathcal{S}$.

A handle-based interface for direct manipulation, allowing the user to specify displacements of surface points $\mathbf{p}_i \in \mathcal{S}$ instead of control points $\mathbf{c}_l$, simplifies the deformation process [Hsu et al. 92]. Given a set of displacement constraints $\mathbf{d}(\mathbf{u}_i) = \bar{\mathbf{d}}_i$ for $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\} = \mathcal{H} \cup \mathcal{F}$, one solves a linear

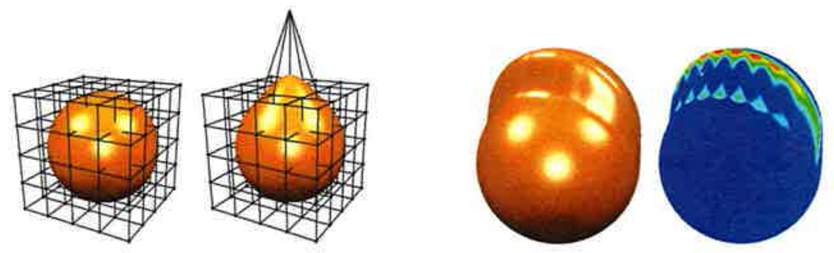


Figure 9.13. In the FFD approach a 3D control grid is used to specify a volumetric displacement function (left). The regular placement of grid basis functions can lead to alias artifacts in the deformed surface (right). (Image taken from [Botsch 05].)

system for the required movements $\boldsymbol{\delta}\mathbf{c}_l$ of control points:

$$\begin{bmatrix} N_1(\mathbf{u}_1) & \ldots & N_n(\mathbf{u}_1) \\ \vdots & \ddots & \vdots \\ N_1(\mathbf{u}_m) & \ldots & N_n(\mathbf{u}_m) \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}\mathbf{c}_1 \\ \vdots \\ \boldsymbol{\delta}\mathbf{c}_n \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{d}}_1 \\ \vdots \\ \bar{\mathbf{d}}_m \end{pmatrix}. \qquad (9.13)$$

This $(m \times n)$ system can be over- as well as under-determined, and is therefore solved using the *pseudo-inverse* [Hsu et al. 92, Golub and Loan 89]. This yields a *least-squares* and *least-norm* solution, which minimizes the error in the constraints $\sum_i \left\|\mathbf{d}(\mathbf{u}_i) - \bar{\mathbf{d}}_i\right\|^2$ as well as the amount of control point movement $\sum_l \left\|\boldsymbol{\delta}\mathbf{c}_l\right\|^2$.

While this yields a well-defined solution, it has two drawbacks: First, in an over-determined setting the displacement constraints cannot be satisfied exactly, but only in the least square sense. Second, in the under-determined setting the remaining degrees of freedom are determined by minimizing control point movements, instead of optimizing for an as-smooth-as-possible deformation, as was the case for the fair surface-based deformation of Section 9.2.

The placement of basis functions $N_l(\mathbf{u})$ on a regular grid is another potential problem. As shown in Figure 9.13 (right), a deformation that is not well-aligned with the grid axes can lead to aliasing artifacts. This problem can be addressed by using more flexible (pre-deformed) control lattices that better represent the desired deformation, but these can be difficult to set up for complex deformations [Coquillart 90, MacCracken and Joy 96].

9.5.2 Cage-Based Freeform Deformation

Cage-based techniques can be considered a generalization of the lattice-based freeform deformation. Instead of a regular control lattice, a so-called *control cage* is used to deform the object. This cage typically is a coarse, arbitrary triangle mesh enclosing the object to be modified, which allows the cage to better match the shape and structure of the embedded object than regular control lattices do (see Figure 9.14).

The vertices $\mathbf{p}_i$ of the original mesh $\mathcal{S}$ can be represented as linear combinations of the cage's control vertices $\mathbf{c}_l$ by

$$\mathbf{p}_i \;=\; \sum_{l=1}^{n} \mathbf{c}_l \, \varphi_l(\mathbf{p}_i)\,, \qquad (9.14)$$

where the weights $\varphi_l(\mathbf{p}_i)$ are *generalized barycentric coordinates* [Floater et al. 05, Ju et al. 05, Ju et al. 07, Lipman et al. 08]. The coordinate functions φ_l in (9.14) therefore correspond to the spline basis functions N_l in Equation (9.12).

Once the per-vertex weights $\varphi_l(\mathbf{p}_i)$ have been pre-computed, the object can be deformed by manipulating the cage vertices $\mathbf{c}_l \mapsto \mathbf{c}_l + \boldsymbol{\delta}\mathbf{c}_l$ and computing the per-vertex displacement as

$$\mathbf{d}(\mathbf{p}_i) \;=\; \sum_{l=1}^{n} \boldsymbol{\delta}\mathbf{c}_l \, \varphi_l(\mathbf{p}_i)\,.$$

Finding control vertex displacements $\boldsymbol{\delta}\mathbf{c}_l$ resulting in a deformation that satisfies user-defined constraints $\mathbf{d}(\mathbf{p}_i) = \bar{\mathbf{d}}_i$ works equivalently to Equation (9.13), with $N_l(\mathbf{u}_i)$ replaced by $\varphi_l(\mathbf{p}_i)$. While being much more flexible

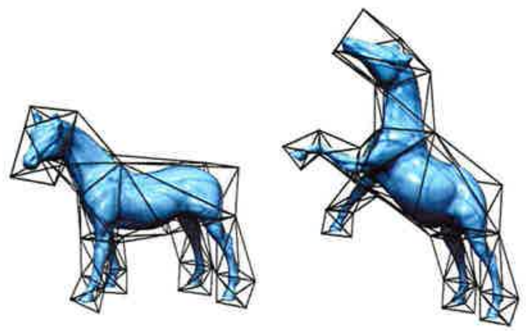


Figure 9.14. Manipulating a horse using a cage-based space deformation, where surface vertices are represented and deformed relative to the cage using generalized barycentric coordinates. (Model courtesy of [Ju et al. 05].)

in terms of the control grid, cage-based methods share the drawback of a least norm solution that does not necessarily correspond to a fair deformation.

9.6 Radial Basis Functions

In the case of surface-based deformations, high-quality results are achieved by interpolating the user's displacement constraints with a deformation function $\mathbf{d}\colon \mathcal{S} \to \mathbb{R}^3$ that minimizes some fairness energies (e.g., Section 9.2). Motivated by this, we derive in this section a smoothly interpolating trivariate *space deformation* function $\mathbf{d}\colon \mathbb{R}^3 \to \mathbb{R}^3$ that minimizes analogous fairness energies.

On a more abstract level, the problem is to find a function $\mathbf{d}$ that interpolates some prescribed values $\bar{\mathbf{d}}_i$ at position $\mathbf{p}_i$, while being smooth and fair in between these constraints. *Radial basis functions* (RBFs) are known to be very well suited for this kind of scattered data interpolation problem [Wendland 05].

A trivariate RBF deformation is defined as a superposition of radially symmetric kernels $\varphi_j(\mathbf{x})$, located at centers $\mathbf{c}_j \in \mathbb{R}^3$ and weighted by $\mathbf{w}_j \in \mathbb{R}^3$:

$$\mathbf{d}(\mathbf{x}) \;=\; \sum_{j=1}^{n} \mathbf{w}_j \, \varphi(\|\mathbf{c}_j - \mathbf{x}\|) \;+\; \boldsymbol{\pi}(\mathbf{x})\,,$$

where $\varphi_j(\mathbf{x}) = \varphi(\|\mathbf{c}_j - \mathbf{x}\|)$ is the basis function corresponding to the jth center $\mathbf{c}_j$, and $\boldsymbol{\pi}(\mathbf{x})$ is a polynomial of low degree used to guarantee polynomial precision. To simplify explanation and notation, we omit the polynomial term in the following.

In order to find an RBF function that interpolates the displacement constraints $\mathbf{d}(\mathbf{p}_i) = \bar{\mathbf{d}}_i$ for $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\} = \mathcal{H} \cup \mathcal{F}$, we use as many RBF kernels as we have constraints and place them on the constraints, i.e., $\mathbf{c}_j = \mathbf{p}_j$. The weights $\mathbf{w}_j$ are then found as the solution of the symmetric linear system

$$\begin{bmatrix} \varphi(\|\mathbf{p}_1 - \mathbf{p}_1\|) & \cdots & \varphi(\|\mathbf{p}_n - \mathbf{p}_1\|) \\ \vdots & \ddots & \vdots \\ \varphi(\|\mathbf{p}_1 - \mathbf{p}_n\|) & \cdots & \varphi(\|\mathbf{p}_n - \mathbf{p}_n\|) \end{bmatrix} \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{d}}_1 \\ \vdots \\ \bar{\mathbf{d}}_n \end{pmatrix}. \qquad (9.15)$$

Once the weights have been computed, i.e., the RBF function $\mathbf{d}$ has been fit to the constraints, the mesh vertices can be displaced as $\mathbf{p}_i' = \mathbf{p}_i + \mathbf{d}(\mathbf{p}_i)$.

The choice of the kernel function φ has a strong influence on the computational complexity and the resulting surface's fairness. While compactly supported radial basis functions lead to sparse linear systems and hence can

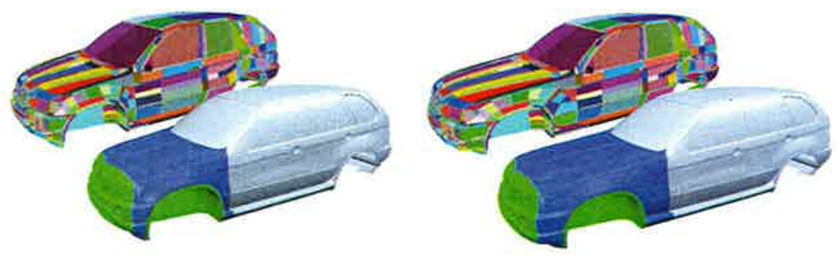


Figure 9.15. Using three independent handles allows one to stretch the car's hood while rigidly preserving the shape of the wheel houses. This 3M triangle model consists of 10k individual connected components, which are neither 2-manifold nor consistently oriented. (Model courtesy of BMW AG. Image taken from [Botsch et al. 06b]. ©2006 ACM, Inc. Included here by permission.)

be used to interpolate a large number of constraints [Morse et al. 01, Ohtake et al. 04], they do not provide the same degree of fairness as basis functions of global support [Carr et al. 01]. It was shown by Duchon [Duchon 77] that the globally supported basis function $\varphi(r) = r^3$ yields a tri-harmonic function $\mathbf{d}$, i.e., $\Delta^3\mathbf{d} = 0$. From variational calculus we know that it therefore minimizes the fairness energy

$$\int_{\mathbb{R}^3} \|\mathbf{d}_{xxx}(\mathbf{x})\|^2 + \|\mathbf{d}_{xxy}(\mathbf{x})\|^2 + \ldots + \|\mathbf{d}_{zzz}(\mathbf{x})\|^2 \,\mathrm{d}\mathbf{x}.$$

Notice that these functions are *conceptually* equivalent to the minimum variation surfaces of [Moreton and Séquin 92] and the tri-harmonic surfaces used in [Botsch and Kobbelt 04a], and therefore provide the same degree of fairness. The difference is that for tri-harmonic RBFs the energy minimization is "built in," whereas for surface-based approaches we explicitly optimized for it (see Section 9.2). The major drawback is that the global support of $\varphi(r) = r^3$ leads to a *dense* linear system (9.15), which is numerically harder to solve (see [Botsch and Kobbelt 05]).

Note that as soon as the constraints change, e.g., by interactively manipulating the handle, the linear system (9.15) has to be solved again for the new right-hand side. For efficiency reasons one can factorize the matrix once, and only compute a back-substitution for each new right-hand side [Golub and Loan 89]. As shown in [Botsch and Kobbelt 05], when restricting to *affine* handle transformations one can precompute special basis functions, which can efficiently be evaluated instead of solving (9.15). Moreover, evaluating these basis functions on the graphics card further accelerates this approach and provides real-time space deformations of several million points per second. As shown in Figure 9.15, even complex surfaces consisting of disconnected patches can be handled by this technique, whereas all surface-based techniques would fail in this situation.

For the two space deformation approaches described in Sections 9.5 and 9.6, the deformed surface S' depends *linearly* on the displacement constraints $\mathbf{d}(\mathbf{p}_i) = \bar{\mathbf{d}}_i$. As a consequence, nonlinear effects such as local detail rotation cannot be achieved, similar to the linear surface-based methods discussed in Sections 9.2–9.4. Although space deformations can be enhanced by multi-scale techniques as well (see, e.g., [Marinov et al. 07]), they generally suffer from the same limitations as surface-based methods when it comes to large-scale deformation, as discussed next.

9.7 Limitations of Linear Methods

The methods we described so far provide high quality results and can be computed robustly and efficiently. However, it is equally important to understand their limitations as it is to understand their strengths. In this section we therefore compare some of the discussed methods and point out their limitations. To this end, the goal is *not* to show the best-possible results each method can produce (those can be found in the original papers) but rather to show under which circumstances each individual method fails. Figure 9.16 shows deformation examples that were particularly chosen to identify the respective limitations of the different techniques. For comparison we show the results of the nonlinear surface deformation PriMo [Botsch et al. 06a], which does not suffer from linearization artifacts.

- The shell-based deformation (Section 9.2), in combination with a multi-scale technique (Section 9.3), works fine for pure translations and yields fair and detail-preserving deformations. However, due to the linearization of the shell energy, this approach fails for large rotations.

- Gradient-based editing (Section 9.4.1) updates the face gradients using the gradient of the handle transformation (its rotation and scale/shear components) and therefore works very well for rotational deformation. However, the explicit propagation of local rotations is translation insensitive, such that the plane example is neither smooth nor detail preserving.

- Laplacian surface editing (Section 9.4.2) implicitly optimizes for local rotations and hence works comparatively well for translations and rotations. However, the required linearization of rotations yields artifacts for large deformations.

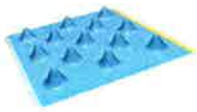
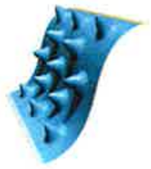
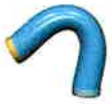
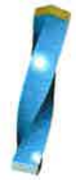
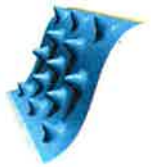
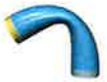
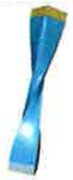
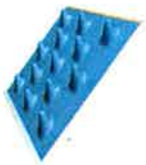
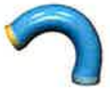
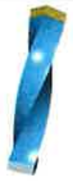
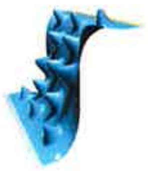
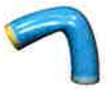
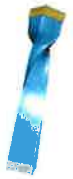
Approach	Pure Translation	120° bend	135° twist
Original models			
Nonlinear deformation [Botsch et al. 06a]			
Shell-based deformation with multi-scale technique [Botsch and Kobbelt 04a] [Botsch et al. 06c]			
Gradient-based editing with harmonic propagation [Zayer et al. 05a]			
Laplacian-based editing with implicit optimization [Sorkine et al. 04]			

Figure 9.16. The extreme examples shown in this comparison matrix were particularly chosen to reveal the limitations of the respective deformation approaches. (Image taken from [Botsch and Sorkine 08]. ©2008 IEEE.)

9.8 Summary and Further Reading

In this chapter we introduced several methods of deforming a given surface and showed that accurate and high-quality deformations can be obtained by minimizing suitable energies, which in the end involve solving a linear system for the deformed vertex positions. With the linear system solvers described in the appendix, this can be computed robustly and at interactive rates. The interested reader can find more details on linear surface deformation methods in [Botsch and Sorkine 08].

However, the accurate physical equations governing the surface deformation process are inherently nonlinear, which requires simplifying or linearizing the involved energies at some point. We have seen the consequence of this in Section 9.7: all linear techniques fail under certain circumstances. The shell-based approach typically works well for translations, but has problems with large rotations, whereas it is the other way around for methods based on differential coordinates. From those examples one can derive the following guidelines for picking the "right" deformation technique for a specific application scenario:

- In technical, CAD-like engineering applications, the required shape deformations are typically rather small, since in many cases an existing prototype has to be adjusted only slightly, but they have high requirements for surface fairness, boundary continuity, and the precise control thereof. For such problems a linearized shell model is typically the best suited.
- In contrast, applications like character animation mostly involve (possibly large) rotations of limbs around bends and joints. Here, methods based on differential coordinates clearly are the better choice. Moreover, the required rotations might be available from, e.g., a sketching interface [Zhou et al. 05, Nealen et al. 05] or a motion capture system [Shi et al. 06].
- Applications that require both large-scale translations and rotations are problematic for all linear approaches. In this case one can either employ a more complex nonlinear technique or split up large deformations into a sequence of smaller ones. While the nonlinear techniques are computationally and implementation-wise more involved, splitting up deformations or providing a denser set of constraints complicates the user interaction.

Thanks to the rapid increase in both computational power and available memory of today's workstations, nonlinear deformation methods have become more and more tractable, which in the last few years has already led to a first set of nonlinear yet interactive surface deformation approaches.

Below we briefly mention some nonlinear approaches for surface-based and space deformation and refer the reader to the original papers. While a nonlinear implementation of the previously discussed approaches seems to be straightforward ("simply do not use any linearization"), in the nonlinear case special attention has to be paid to computational efficiency and numerical robustness of the involved energy minimizations.

Nonlinear surface deformation. Pyramid coordinates (see [Sheffer and Kraevoy 04, Kraevoy and Sheffer 06]) can be considered nonlinear versions of Laplacian coordinates, leading to differential coordinates invariant under rigid motions, which can be used for deformation as well as for morphing.

Huang et al. employ a nonlinear version of the volumetric graph Laplacian, which also features nonlinear volume preservation constraints [Huang et al. 06]. In order to increase the performance and efficiency of their optimization, they use a subspace approach: the original mesh is embedded in a coarse control cage (see Section 9.5.2), and the optimization is performed on the cage vertices $\mathbf{c}_j$ while considering the constraints from the original mesh vertices $\mathbf{p}_i$ in a least-squares manner.

An alternative approach to subspace methods is the handle-aware isoline technique of [Au et al. 07]. In a preprocessing step one constructs a set of iso-lines of the geodesic distance from either the fixed regions or the handle regions, similar in spirit to [Zayer et al. 05a]. For each of these iso-lines, a local transformation $\mathbf{M}_i$ for a Laplacian-based deformation is found by a nonlinear optimization. The number of required iso-lines is relatively small, which guarantees an efficient numerical optimization and thereby allows for interactive editing.

Shi et al. combine Laplacian-based deformation with skeleton-based inverse kinematics [Shi et al. 07]. Their approach allows for easy and intuitive character posing, featuring control of lengths, rigidity, and joint limits, but it in turn requires a complex cascading optimization for the involved nonlinear energy minimization.

PriMo [Botsch et al. 06a] is a nonlinear version of the shell-based minimization of bending and stretching energies. The surface is modeled as a thin layer of triangular prisms, which are coupled by a nonlinear elastic energy. During deformation the prisms are kept rigid, which allows for a robust geometric optimization. A hierarchical optimization is used to increase the computational efficiency.

The as-rigid-as-possible surface deformation of [Sorkine and Alexa 07] models local rotations in terms of each vertex's one-ring. An easy-to-implement alternating optimization solves for the local rotations and the new vertex positions.

Eigensatz and Pauly introduce a surface deformation method that allows directly prescribing positional, metric, and curvature constraints anywhere on the surface. A global nonlinear optimization solves for a deformed surface that satisfies these user constraints as best as possible, while minimizing the overall metric and curvature distortion [Eigensatz and Pauly 09].

Nonlinear space deformation. Sumner et al. compute detail-preserving space deformations by formulating an energy functional that explicitly penalizes deviation from local rigidity by optimizing the local deformation gradients to be rotations [Sumner et al. 07]. In addition to static geometries, their method can also be applied to hand-crafted animations and precomputed simulations.

Botsch et al. extend the PriMo framework [Botsch et al. 06a] to deformations of solid objects [Botsch et al. 07]. The input model is voxelized in an adaptive manner, and the resulting hexahedral cells are kept rigid under deformations to ensure numerical robustness. The deformation is governed by a nonlinear elastic energy coupling neighboring rigid cells.

Another class of approaches uses divergence-free vector fields to deform shapes [Angelidis et al. 06, von Funck et al. 06]. The advantage of those techniques is that by construction they yield volume-preserving and intersection-free deformations. As a drawback, it is harder to construct vector fields that exactly satisfy user-defined deformation constraints.