Interactive Shape Modeling and Deformation

Olga Sorkine[1] and Mario Botsch[2]

[1]Courant Institute of Mathematical Sciences, New York University
[2]Graphics & Geometry Group, Bielefeld University

Abstract

We present a tutorial that covers the latest research advances in interactive 3D shape modeling and deformation, a highly relevant topic for CAGD, engineering applications, and computer animation for movies or games. We focus on the essence of the underlying theory and principles, as well as the practical aspects of algorithm design and development involved in interactive shape editing. Our presentation is meant to be comparative*, juxtaposing various recently proposed approaches and revealing their pros and cons in different modeling scenarios. As such, our class is intended for both researchers and practitioners, helping to sift through the large body of work on interactive modeling by a systematic, hands-on overview.*

1. Introduction

Modeling, editing, and deformation of 3D shapes is an essential part of the digital content creation process. It is a very challenging research field, since complex mathematical formulations (i) have to be hidden behind an intuitive user interface and (ii) have to be implemented in a sufficiently efficient and robust manner to allow for interactive applications.

We put together the latest research progress in the flourishing field of interactive shape modeling, and present it in a way accessible to scientists and practitioners. Our goal is to deliver a clear and systematic survey of the underlying theory, algorithms, and principles that enable new shape modeling paradigms and simplify previously complex and time-consuming shape modeling and editing tasks. We present and compare recent advances in 3D surface modeling and deformation, discuss related user interface issues, and emphasize practical considerations for researchers and developers.

We start by briefly reviewing shape modeling preliminaries, namely the shape representations used in geometric modeling (Section 2) and the basic differential geometry concepts that are used throughout the tutorial (Section 3). We then discuss linear surface-based deformation methods in Section 4, roughly classified into (multiresolution) bending energy minimization (Section 4.3 and Section 4.4) and differential coordinates (Section 4.5). Linear space deformations are then presented in Section 5.

By "linear" we mean that the discussed techniques employ global linear optimization and/or only local nonlinear computations. We point out the inherent limitations of linear(ized) deformation approaches (Section 4.6), which are avoided by fully nonlinear techniques. The surface-based nonlinear deformation is addressed in Section 6, while nonlinear space deformation methods are discussed in Section 7.

2. Shape Representations

We start by briefly reviewing the shape representations typically used in geometric modeling. We focus on explicit surface representations, which can roughly be classified as the range of a bivariate parameterization function. We first describe spline surfaces in Section 2.1, since those are the main representation used in most CAD system. Subdivision surfaces generalize splines surfaces and are much more flexible in representing complex shapes (Section 2.2).

2.1. Spline Surfaces

Tensor-product spline surfaces are the standard surface representation of today's CAD systems. They are used for constructing high-quality surfaces from scratch as well as for later surface deformation tasks. Spline surfaces can conveniently be described by the B-spline basis functions $N_i^n(\cdot)$, for more detail see [Far97].

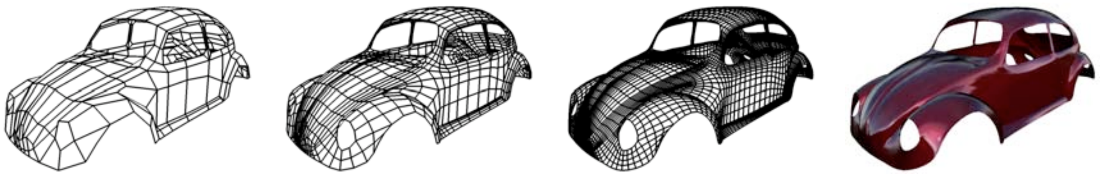


Figure 1: *Subdivision surfaces are generated by an iterative refinement of a coarse control mesh.*

A tensor product spline surface $\mathbf{f}$ of degree n is a piecewise polynomial surface that is built by connecting several polynomial patches in a smooth C^{n-1} manner:

$$\mathbf{f}(u,v) \;=\; \sum_{i=0}^{m}\sum_{j=0}^{m} \mathbf{c}_{ij} N_i^n(u) N_j^n(v) \;\; .$$

The *control points* $\mathbf{c}_{ij} \in \mathbb{R}^3$ define the so-called *control grid* of the spline surface. The scalar-valued basis functions satisfy $N_i^n(u) \geq 0$ and $\sum_i N_i^n \equiv 1$. Hence, each surface point $\mathbf{f}(u,v)$ is a convex combination of the control points $\mathbf{c}_{ij}$, i.e., the surface lies within the convex hull of the control grid. Due to the small support of the basis functions, each control point has local influence only. These two properties cause spline surfaces to closely follow the control grid, thereby providing a geometrically intuitive metaphor for modeling surfaces by adjusting its control points.

A tensor-product surface — as the image of a rectangular domain under the parameterization $\mathbf{f}$ — always represents a rectangular surface patch embedded in $\mathbb{R}^3$. If shapes of more complicated topological structure are to be represented by spline surfaces, the model has to be decomposed into a large number of (possibly trimmed) tensor-product patches.

As a consequence of these *topological constraints*, typical CAD models consist of a huge collection of surface patches. In order to represent a high quality, globally smooth surface, these patches have to be connected in a smooth manner, leading to additional *geometric constraints*, that have to be taken care of throughout all surface processing phases. The large number of surface patches and the resulting topological and geometric constraints significantly complicate surface construction and in particular the later surface modeling tasks.

2.2. Subdivision Surfaces

Subdivision surfaces [ZSD*00] can be considered as a generalization of spline surfaces, since they are also controlled by a coarse *control mesh*, but in contrast to spline surfaces they can represent surfaces of arbitrary topology. Subdivision surfaces are generated by repeated refinement of control meshes: After each topological refinement step, the positions of the (old and new) vertices are adjusted based on a set of local averaging rules. A careful analysis of these rules reveals that in the limit this process results in a surface of provable smoothness (cf. Fig. 1).

As a consequence, subdivision surfaces are restricted neither by topological nor by geometric constraints as spline surfaces are, and their inherent hierarchical structure allows for highly efficient algorithms. However, subdivision techniques are restricted to surfaces with so-called semi-regular *subdivision connectivity*, i.e., surface meshes whose triangulation is the result of repeated refinement of a coarse control mesh. As this constraint is not met by arbitrary surfaces, those would have to be *remeshed* to subdivision connectivity in a preprocessing step [EDD*95, LSS*98, KVLS99, GVSS00]. But as this remeshing corresponds to a resampling of the surface, it usually leads to sampling artifacts and loss of information. In order to avoid the restrictions caused by these *connectivity constraints*, we propose to work on arbitrary triangle meshes, since they provide higher flexibility and also allow for efficient surface processing.

2.3. Triangle Meshes

A triangle mesh $\mathcal{M}$ consists of a geometric and a topological component, where the latter can be represented by a graph structure (simplicial complex) with a set of vertices

$$\mathcal{V} = \{v_1, \ldots, v_V\}$$

and a set of triangular faces connecting them

$$\mathcal{F} = \{f_1, \ldots, f_F\}, \quad f_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V} \, .$$

The connectivity of a triangle mesh can also be represented in terms of the edges of the respective graph

$$\mathcal{E} = \{e_1, \ldots, e_E\}, \quad e_i \in \mathcal{V} \times \mathcal{V} \;\; .$$

The geometric embedding of a triangle mesh in $\mathbb{R}^3$ is specified by associating a 3D position $\mathbf{x}_i$ to each vertex $v_i \in \mathcal{V}$:

$$\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_V\}, \quad \mathbf{x}_i := \mathbf{x}(v_i) = \begin{pmatrix} x(v_i) \\ y(v_i) \\ z(v_i) \end{pmatrix} \in \mathbb{R}^3 \;\; .$$

Each face $f \in \mathcal{F}$ then represents a triangle in 3-space specified by its three vertex positions.

If a sufficiently smooth surface is approximated by such a piecewise linear function, the approximation error is of the order $O(h^2)$, with h denoting the maximum edge length. Due to this quadratic approximation power, the error is reduced by a factor of $1/4$ when halving the edge lengths. As this refinement splits each triangle into four sub-triangles, it increases the number of triangles from F to $4F$ (cf. Fig. 2).

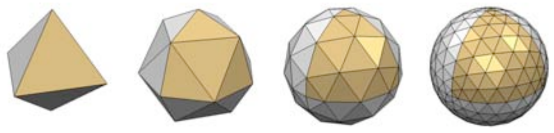


Figure 2: *Each subdivision step halves the edge lengths, increases the number of faces by a factor of 4, and reduces the error by a factor of* $\frac{1}{4}$.

Hence, the approximation error of a triangle mesh is inversely proportional to the number of its faces. The actual magnitude of the approximation error depends on the second order terms of the Taylor expansion, i.e., on the curvature of the underlying smooth surface. From this we can conclude that a sufficient approximation is possible with just a moderate mesh complexity: The vertex density has to be locally adapted to the surface curvature, such that flat areas are sparsely sampled, while in curved regions the sampling density is higher.

An important topological quality of a surface is whether or not it is *two-manifold*, which is the case if for each point the surface is locally homeomorphic to a disk (or a half-disk at boundaries). A triangle mesh is two-manifold, if it does neither contain non-manifold edges or non-manifold vertices, nor self-intersections. A *non-manifold edge* has more than two incident triangles and a *non-manifold vertex* is generated by pinching two surface sheets together at that vertex, such that the vertex is incident to two fans of triangles (cf. Fig. 3). Non-manifold meshes are problematic for most algorithms, since around non-manifold configurations there exists no well-defined local geodesic neighborhood.

For piecewise (polynomial) surface definitions, the most difficult part is the construction of smooth transitions between neighboring patches. Since for triangle meshes, we only require C^0 continuity, we only have to make sure that neighboring faces share a common edge (two common vertices). This makes polygon meshes the most simple and flexible continuous surface representation.

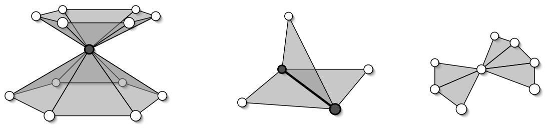


Figure 3: *Two surface sheets meet at a non-manifold vertex (*left*). A non-manifold edge has more than two incident faces (*center*). The right configuration, although being non-manifold in the strict sense, can be handled by most data structures.*

3. Differential Geometry

This section provides a brief review of important concepts from differential geometry that form the basis of the definition of the discrete operators on triangle meshes. For an in-depth discussion of continuous and discrete differential geometry we refer the reader to the textbook [dC76] and the course notes [GDP*05], respectively.

3.1. Continuous Differential Geometry

Let us consider continuous surface $\mathcal{S} \subset \mathbb{R}^3$ that is given in parametric form as

$$\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))^T \quad ,$$

where the coordinate functions x, y, and z are (sufficiently often) differentiable functions in u and v. The partial derivatives w.r.t. u and v are denoted by

$$\mathbf{x}_u := \frac{\partial \mathbf{x}}{\partial u} \quad \text{and} \quad \mathbf{x}_v := \frac{\partial \mathbf{x}}{\partial v}$$

and span the tangent plane at $\mathbf{x}(u,v) \in \mathcal{S}$. The normal vector perpendicular to this tangent plane is given as

$$\mathbf{n} \;=\; \frac{\mathbf{x}_u \times \mathbf{x}_v}{\|\mathbf{x}_u \times \mathbf{x}_v\|} \quad .$$

The *first fundamental form* $\mathbf{I}$ of $\mathbf{x}$ is given by the matrix

$$\mathbf{I} \;:=\; \begin{pmatrix} \mathbf{x}_u^T\mathbf{x}_u & \mathbf{x}_u^T\mathbf{x}_v \\ \mathbf{x}_u^T\mathbf{x}_v & \mathbf{x}_v^T\mathbf{x}_v \end{pmatrix} \quad , \tag{1}$$

which defines an inner product on the tangent space of $\mathcal{S}$. The *second fundamental form* $\mathbf{II}$ is defined as

$$\mathbf{II} \;:=\; \begin{pmatrix} \mathbf{x}_{uu}^T\mathbf{n} & \mathbf{x}_{uv}^T\mathbf{n} \\ \mathbf{x}_{uv}^T\mathbf{n} & \mathbf{x}_{vv}^T\mathbf{n} \end{pmatrix} \quad . \tag{2}$$

The symmetric bilinear first and second fundamental forms allow to measure length, angles, area, and curvatures on the surface.

Let $\mathbf{t} = a\mathbf{x}_u + b\mathbf{x}_v$ be a unit vector in the tangent plane at $\mathbf{x}$, represented as $\bar{\mathbf{t}} = (a,b)^T$ in the local coordinate system. The *normal curvature* $\kappa_n(\bar{\mathbf{t}})$ is the curvature of the planar curve that results from intersecting $\mathcal{S}$ with the plane through $\mathbf{x}$ spanned by $\mathbf{n}$ and $\mathbf{t}$. The normal curvature in direction $\bar{\mathbf{t}}$ can be expressed in terms of the fundamental forms as

$$\kappa_n(\bar{\mathbf{t}}) \;=\; \frac{\bar{\mathbf{t}}^T\,\mathbf{II}\,\bar{\mathbf{t}}}{\bar{\mathbf{t}}^T\,\mathbf{I}\,\bar{\mathbf{t}}} \quad .$$

The minimal normal curvature κ_1 and the maximal normal curvature κ_2 are called *principal curvatures*. The associated tangent vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ are called *principal directions* and are always perpendicular to each other.

The *Gaussian curvature* K is defined as the product of the principal curvatures, i.e.,

$$K \;=\; \kappa_1\kappa_2, \tag{3}$$

the *mean curvature* H as the average of the principal curvatures, i.e.,

$$H = \frac{\kappa_1 + \kappa_2}{2}. \tag{4}$$

The following sections will make extensive use of the *Laplace operator* Δ, respectively, the *Laplace-Beltrami operator* $\Delta_{\mathcal{S}}$. In general, the Laplace operator is defined as the divergence of the gradient, i.e.

$$\Delta = \nabla^2 = \nabla \cdot \nabla \ .$$

In Euclidean space this second order differential operator can be written as the sum of second partial derivatives

$$\Delta f = \operatorname{div} \nabla f = \sum_i \frac{\partial^2 f}{\partial x_i^2} \tag{5}$$

with Cartesian coordinates x_i. The *Laplace-Beltrami operator* extends this concept to functions defined on surfaces. For a given function f defined on a manifold surface $\mathcal{S}$ the Laplace-Beltrami is defined as

$$\Delta_{\mathcal{S}} f = \operatorname{div}_{\mathcal{S}} \nabla_{\mathcal{S}} f \ ,$$

which requires a suitable definition of the divergence and gradient operators on manifolds (see [dC76] for details). Applied to the coordinate function $\mathbf{x}$ of the surface the Laplace-Beltrami operator evaluates to the mean curvature normal

$$\Delta_{\mathcal{S}} \mathbf{x} = -2H\mathbf{n} \ .$$

3.2. Discrete Differential Geometry

The deformation approaches described later make extensive use of the Laplace-Beltrami operator. The operator, however, requires the existence of second derivatives. While this is satisfied for smooth spline surfaces, it is not true for piecewise linear triangle meshes, such that the concepts introduced above cannot be applied directly.

The following definitions of *discrete* differential operators are based on the assumption that meshes can be interpreted as piecewise linear approximations of smooth surfaces. The general idea of the techniques described below is to compute discrete differential properties as spatial averages over a local neighborhood $\mathcal{N}(\mathbf{x})$ of a point $\mathbf{x}$ on the mesh. For our purposes, $\mathbf{x}$ coincides with a mesh vertex v_i, and one-ring neighborhoods $\mathcal{N}_1(v)$ are used as averaging domain.

Given a piecewise linear scalar function $f : \mathcal{S} \to \mathbb{R}$ defined on the mesh $\mathcal{S}$, its discrete Laplace-Beltrami at a vertex v_i can be discretized in the form

$$\Delta_{\mathcal{S}} f(v_i) = w_i \sum_{v_j \in \mathcal{N}_1(v_i)} w_{ij} \left(f(v_j) - f(v_i) \right), \tag{6}$$

where $v_j \in \mathcal{N}_1(v_i)$ are the incident one-ring neighbors of v_i (cf. Fig. 4). The discretization depends on the per-vertex normalization weights w_i and the edge weights $w_{ij} = w_{ji}$.

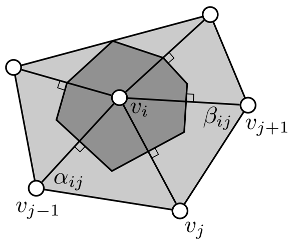


Figure 4: *The angles α_{ij} and β_{ij} and the (dark grey) Voronoi area A_i used to discretize the Laplace-Beltrami $\Delta_{\mathcal{S}}$ at a vertex v_i in equations (6) and (7).*

There exist several variations of the weights used in the typically employed Laplacian discretization (6). The uniform Laplacian, proposed by [Tau95], uses the weights

$$w_{ij} = 1 \ , \quad w_i = \frac{1}{\sum_j w_{ij}}.$$

Since this discretization takes neither edge lengths nor angles into account, it cannot provide a good approximation for irregular meshes. Better results can be achieved by

$$w_{ij} = \frac{1}{2} \left(\cot \alpha_{ij} + \cot \beta_{ij} \right) \ , \quad w_i = 1,$$

which now considers angles, but not varying vertex densities [YZX*04]. The best results are obtained by including the per-vertex normalization weights

$$w_{ij} = \frac{1}{2} \left(\cot \alpha_{ij} + \cot \beta_{ij} \right) \ , \quad w_i = \frac{1}{A_i}, \tag{7}$$

where α_{ij} and β_{ij} are the two angles opposite to the edge (v_i, v_j), and A_i is the Voronoi area of vertex v_i. The latter is defined in [MDSB03] as the area of the surface region built by connecting incident edges' midpoints with triangle circumcenters (for acute triangles) or midpoints of opposite edges (for obtuse triangles), as shown in Fig. 4.

This is the de-facto standard discretization, as proposed by [PP93, DMSB99, MDSB03] and employed for instance in [BK04a]. A qualitative comparison of the three discretizations is given in Fig. 5; in this example curvature energies are minimized by solving $\Delta_{\mathcal{S}}^2 \mathbf{x} = \mathbf{0}$, since smooth surfaces are visually easier to evaluate than smooth deformations. A more detailed analysis of different discretizations can be found in [GDP*05, WMKG07].

While the cotangent discretization clearly gives the best results, it can also lead to numerical problems in the presence of near-degenerate triangles, since then the cotangent values degenerate and the resulting matrices become singular. In this case the degenerate triangles would have to be eliminated [BK01] in a preprocess, or even the whole surface has to be remeshed [AUGA05].

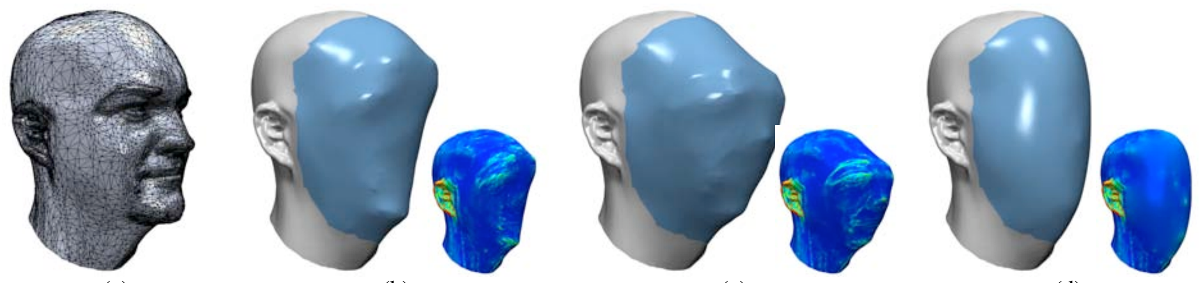


Figure 5: *Comparison of different Laplace-Beltrami discretizations when solving* $\Delta^2_{\mathcal{S}}\mathbf{x} = \mathbf{0}$*. (a) irregular triangulation of the input mesh, (b) uniform Laplacian, (c) cotangent Laplacian without the area term, (d) cotangent discretization including the per-vertex normalization. The small images visualize the respective mean curvatures.*

Higher-order Laplacians can then be defined recursively

$$\Delta^k_{\mathcal{S}} f(v_i) \;=\; w_i \sum_{v_j \in \mathcal{N}_1(v_i)} w_{ij} \left(\Delta^{k-1}_{\mathcal{S}} f(v_j) - \Delta^{k-1}_{\mathcal{S}} f(v_i) \right) \;.$$

After this brief review of surface representations and differential geometry concepts we can now focus on shape deformation, starting with *linear* deformation approaches.

4. Linear Surface-Based Deformation

In the case of *surface-based* deformations we are looking for a displacement function $\mathbf{d} : \mathcal{S} \to \mathbb{R}^3$ that lives *on the surface*. This function maps each point of the given surface $\mathcal{S}$ to its deformed version $\mathcal{S}'$:

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

In particular in engineering applications, exact control of the deformation process is crucial, i.e., one has to be able to specify displacements for a set of constrained points $\mathcal{C}$:

$$\mathbf{d}(\mathbf{x}_i) = \mathbf{d}_i \,, \quad \forall \mathbf{x}_i \in \mathcal{C} \;.$$

Since we are targetting interactive shape deformations, another important aspect is the amount of user interaction required to specify the desired deformation function $\mathbf{d}$.

We will first discuss two standard deformation approaches, namely spline surfaces (Section 4.1) and transformation propagation (Section 4.2), before turning to the more flexible and more powerful variational techniques (Sections 4.3 – 4.5).

4.1. Tensor-Product Spline Surfaces

As discussed in Section 2.1, the traditional surface representation in CAGD are spline surfaces. They are controlled by an intuitive control point metaphor and yield high quality smooth surfaces. A single tensor-product spline patch is defined as

$$\mathbf{f}(u,v) \;=\; \sum_{i=0}^{m} \sum_{j=0}^{m} \mathbf{c}_{ij} N^n_i(u) N^n_j(v) \;.$$

Each control point $\mathbf{c}_{ij}$ is associated with a basis function

$$N_{ij}(u,v) \;:=\; N^n_i(u) N^n_j(v) \;.$$

A translation of a control point $\mathbf{c}_{ij}$ therefore adds a smooth bump of rectangular support to the surface (cf. Fig. 6, left). Every more sophisticated modeling operation has to be composed from such smooth elementary modifications. A general spline displacement function has the form

$$\mathbf{d}(u,v) \;=\; \sum_{i=0}^{m} \sum_{j=0}^{m} \delta\mathbf{c}_{ij} N_{ij}(u,v) \;\;,$$

where $\delta\mathbf{c}_{ij}$ denotes the change of the control point $\mathbf{c}_{ij}$. The support of the deformation is the union of the supports of individual basis functions. As the positions of the basis functions are fixed to the initial grid of control points, this prohibits a fine-grained control of the desired support region. Moreover, the composition of fixed basis functions located on a fixed grid might lead to alias artifacts in the resulting surface, as shown in Fig. 6.

It was also shown in Section 2.1 that tensor-product spline surfaces are restricted to rectangular domains, and that complex surfaces therefore have to be composed by a large number of spline patches. Specifying complex deformations in terms of control point movements thus involves a lot of user interaction, since smoothness constraints across patch boundaries have to be considered during the deformation process.

Also note that prescribing constraints $\mathbf{d}(u_i, v_i) = \mathbf{d}_i$ requires to solve a linear system for the control point displacements $\delta\mathbf{c}_{ij}$. These systems can be over- as well as under-determined, and hence are typically solved by least squares and least norm techniques. However, in the first case, the system cannot be solved exactly, and in the latter case the minimization of control point displacements does not necessarily lead to fair deformations, which would require to minimize some fairness energy (Section 4.3).

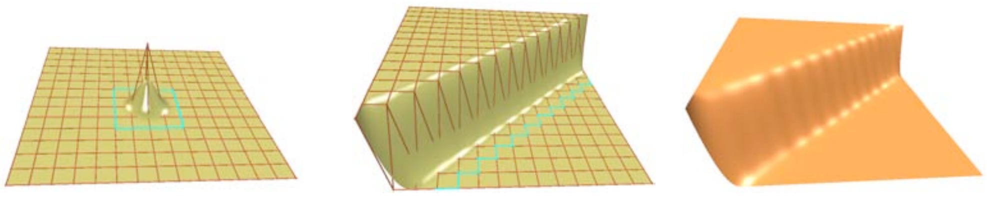


Figure 6: *A modeling example using a bi-cubic tensor-product spline surface. Each control point is associated with a smooth basis function of fixed rectangular support (*left*). This fixed support and the fixed regular placement of the control points, resp. basis functions, prevents a precise support specification (*center*) and can lead to alias artifacts in the resulting surface, that are revealed by more sensitive surface shading (*right*).*

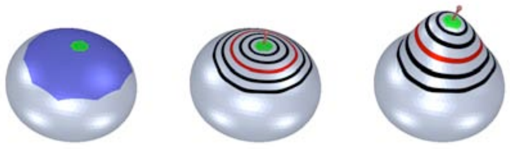


Figure 7: *After specifying the blue support region and the green handle region (*left*), a smooth scalar field is constructed that is* 1 *at the handle and* 0 *outside of the support (*center*). This scalar field is used to propagate and damp the handle's transformation (*right*).*

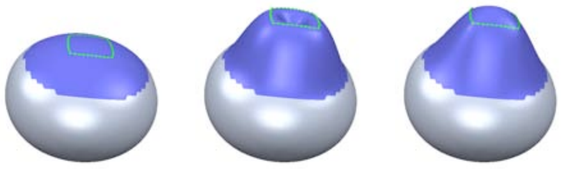


Figure 8: *A sphere is deformed by lifting a handle polygon (*left*). Propagating this translation based on geodesic distance causes a dent in the interior of the handle polygon (*center*). The more intuitive solution of a smooth interpolation (*right*) cannot be achieved with this approach; it was produced by variational energy minimization (Section 4.3).*

4.2. Transformation Propagation

The main drawback of spline-based deformations is that the underlying mathematical surface representation is identical to the basis functions that are used for the surface deformation. To overcome this limitation, the deformation basis functions consequently should be independent of the actual surface representation.

A popular approach falling into this category works as follows (cf. Fig. 7): In a first step the user specifies the support of the deformation (the region which is allowed to change) and a handle region $\mathcal{H}$ within it. The handle region is directly deformed using any modeling interface, and its transformation is smoothly interpolated within the support region in order to blend between the transformed handle $\mathcal{H}$ and the fixed part $\mathcal{F}$ of the surface.

This smooth blend is controlled by a scalar field $s : \mathcal{S} \to [0,1]$, which is 1 at the handle (full deformation), 0 outside the support (no deformation), and smoothly blends between 1 and 0 within the support region. One way to construct such a scalar field is to compute geodesic (or Euclidean) distances $\mathrm{dist}_{\mathcal{F}}(\mathbf{x})$ and $\mathrm{dist}_{\mathcal{H}}(\mathbf{x})$ from $\mathbf{x}$ to the fixed part $\mathcal{F}$ and the handle region $\mathcal{H}$, respectively, and to define [BK03a, PKKG03]

$$s(\mathbf{x}) = \frac{\mathrm{dist}_{\mathcal{F}}(\mathbf{x})}{\mathrm{dist}_{\mathcal{F}}(\mathbf{x}) + \mathrm{dist}_{\mathcal{H}}(\mathbf{x})} .$$

While this definition of the scalar field is continuous, it might not be smooth for concave handle shapes. An alternative is to construct the function $s(\mathbf{x})$ to be smooth and harmonic, i.e. $\Delta s = 0$ [ZRKS05]. Using the Laplace-Beltrami discretization of Section 3.2, this leads to a linear system to be solved for the values of s at the unconstrained vertices:

$$\begin{aligned} \Delta_{\mathcal{S}} s(\mathbf{x}_i) &= 0, & \mathbf{x}_i &\notin \mathcal{H} \cup \mathcal{F}, \\ s(\mathbf{x}_i) &= 1, & \mathbf{x}_i &\in \mathcal{H}, \\ s(\mathbf{x}_i) &= 0, & \mathbf{x}_i &\in \mathcal{F}. \end{aligned} \tag{8}$$

Both types of scalar fields can further be enhanced by a transfer function $t(s(\mathbf{x}))$, which provides more control of the blending process. The damping of the handle transformation is then performed separately on the rotation, scale/shear, and translation components, for instance like in [PKKG03]. In cases where these individual components of the transformation are not given, they can be computed by polar decomposition [SD92].

As shown in Fig. 8, the major problem with this approach is that the distance-based propagation of transformations will typically not result in the geometrically most intuitive solution. This would require the interpolation of the handle transformation by the displacement function $\mathbf{d}$, while otherwise minimizing some fairness energies.

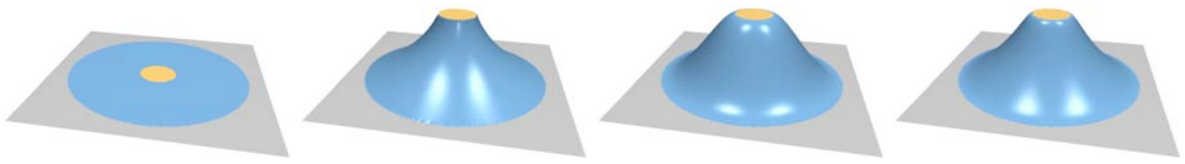


Figure 9: *The surface $\mathcal{S}$ (*left*) is edited by minimizing its deformation energy subject to user-defined constraints that fix the gray part $\mathcal{F}$ of the surface and prescribe the transformation of the yellow handle region $\mathcal{H}$. The deformation energy (10) consists of stretching and bending terms, and the examples show pure stretching with $k_s = 1$, $k_b = 0$ (*center left*), pure bending with $k_s = 0$, $k_b = 1$ (*center right*), and a weighted combination with $k_s = 1$, $k_b = 10$ (*right*).*

4.3. Variational Energy Minimization

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

As mentioned in Section 3, the first and second fundamental forms $\mathbf{I}(u,v)$ and $\mathbf{II}(u,v)$ can be used to measure geometrically intrinsic properties of $\mathcal{S}$, such as lengths, areas, and curvatures. When the surface $\mathcal{S}$ is deformed to $\mathcal{S}'$, and 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 [TPBF87]:

$$E_{\text{shell}}(\mathcal{S}') = \int_\Omega k_s \left\|\mathbf{I}' - \mathbf{I}\right\|^2 + k_b \left\|\mathbf{II}' - \mathbf{II}\right\|^2 \,\mathrm{d}u\mathrm{d}v. \quad (9)$$

The stiffness parameters k_s and k_b are used to control the resistance to stretching and bending, respectively. In a modeling application one would have to minimize the elastic energy (9) subject to user-defined deformation constraints. As shown in Fig. 9, this typically means fixing certain surface parts $\mathcal{F} \subset \mathcal{S}$ and prescribing displacements for the so-called *handle* region(s) $\mathcal{H} \subset \mathcal{S}$.

However, this nonlinear minimization is computationally too expensive for interactive applications. It is therefore simplified and linearized by replacing the difference of fundamental forms by partial derivatives of the displacement function $\mathbf{d}$ [CG91, WW92]:

$$\tilde{E}_{\text{shell}}(\mathbf{d}) = \int_\Omega k_s \left(\|\mathbf{d}_u\|^2 + \|\mathbf{d}_v\|^2\right) + k_b \left(\|\mathbf{d}_{uu}\|^2 + 2\|\mathbf{d}_{uv}\|^2 + \|\mathbf{d}_{vv}\|^2\right) \mathrm{d}u\mathrm{d}v \ , \quad (10)$$

where we again used the notation $\mathbf{d}_x = \frac{\partial \mathbf{d}}{\partial x}$ and $\mathbf{d}_{xy} = \frac{\partial^2 \mathbf{d}}{\partial x \partial y}$.

For the efficient minimization of (10) we apply variational calculus, which yields the corresponding *Euler-Lagrange* equations that characterize the minimizer of (10), again subject to user constraints:

$$-k_s \Delta_{\mathcal{S}} \mathbf{d} + k_b \Delta_{\mathcal{S}}^2 \mathbf{d} = \mathbf{0}. \quad (11)$$

Notice that for the second derivatives in (10) to closely approximate surface curvatures (i.e., bending), the parameterization $\mathbf{x} : \Omega \to \mathcal{S}$ should be as close to isometric as possible. Therefore Ω is typically chosen to equal $\mathcal{S}$, such that $\mathbf{d} : \mathcal{S} \to \mathbb{R}^3$ is defined on the manifold $\mathcal{S}$ itself. As a consequence, the Laplace operator in (11) corresponds to the Laplace-Beltrami operator (see Section 3).

The order k of partial derivatives in the energy (10) or in the corresponding Euler-Lagrange equations $(-1)^k \Delta_{\mathcal{S}}^k \mathbf{d} = 0$ defines the maximum continuity C^{k-1} for interpolating displacement constraints [BK04a]. Hence, minimizing (10) by solving (11) provides C^1 continuous surface deformations, as can also be observed in Fig. 9. On a discrete triangle mesh, the C^1 constraints are defined by the first two rings of fixed vertices $\mathcal{F}$ and handle vertices $\mathcal{H}$.

Using the cotangent discretization of the Laplace-Beltrami defined in Section 3.2, the Euler-Lagrange PDE (11) turns into a sparse bi-Laplacian linear system:

$$\begin{aligned} -k_s \Delta_{\mathcal{S}} \mathbf{d} + k_b {\Delta_{\mathcal{S}}}^2 \mathbf{d} &= \mathbf{0}, & \mathbf{x}_i &\notin \mathcal{H} \cup \mathcal{F}, \\ \mathbf{d}(\mathbf{x}_i) &= \mathbf{d}_i, & \mathbf{x}_i &\in \mathcal{H}, \\ \mathbf{d}(\mathbf{x}_i) &= \mathbf{0}, & \mathbf{x}_i &\in \mathcal{F}. \end{aligned} \quad (12)$$

Interactively manipulating the handle region $\mathcal{H}$ changes the boundary constraints of the optimization, i.e., the right-hand side of the linear system Eq. (12). As a consequence, this system has to be solved in each frame. In Section A we will discuss efficient linear system solvers that are particularly suited for this multiple right-hand side problem. Also notice that restricting to *affine* transformation of the handle region $\mathcal{H}$ (which is usually sufficient) allows to precompute basis functions of the deformation, such that instead of solving (12) in each frame, only the basis functions have to be evaluated [BK04a].

The approaches of [KCVS98] and [BK04a] can be considered as instances of the framework described in this section, since both methods solve bi-Laplacian system to derive fair shape deformations. Other methods are conceptually similar, but achieve smooth deformations, for instance by hierarchical smoothing [GSS99] or subdivision surfaces [ZSS97].

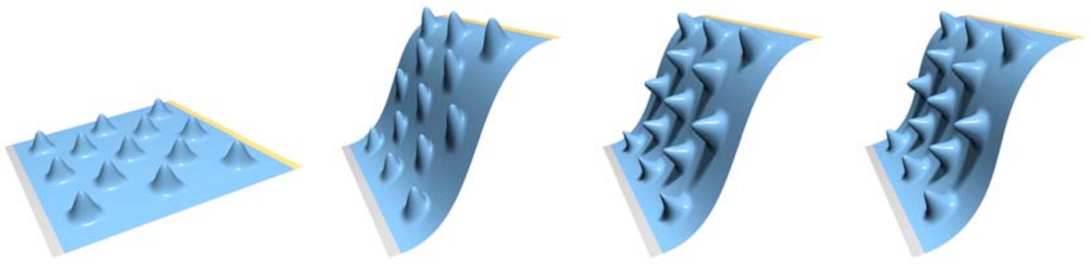


Figure 10: *The right strip $\mathcal{H}$ of the bumpy plane (*left*) is lifted. The intuitive local rotations of geometric details cannot be achieved by a linearized deformation alone (*center left*). A multiresolution approach based on normal displacements (*center right*) correctly rotates local details, but also distortions them, which can be seen in the left-most row of bumps. The more accurate result of a nonlinear technique is shown on the* right*.*

4.4. Multiresolution Deformation

The variational optimization techniques introduced in the last section provide C^1 continuous, smooth, and fair surface deformations. Interactive performance is achieved by simplifying or linearizing the nonlinear shell energy (9), such that the techniques become linear in the sense that they only require solving a linear system for the deformed surface $\mathcal{S}'$.

However, as a consequence of this linearization, such methods typically do not correctly handle fine-scale surface details, as depicted in Fig. 10. The local rotation of geometric details is an inherently nonlinear behavior, and hence cannot be modeled by purely linear techniques. One way to preserve geometric details under global deformations, while still using a linear deformation approach, is to use *multiresolution* techniques, as described in this section.

Multiresolution (or multi-scale) techniques perform a frequency decomposition of the object in order to provide global deformations with intuitive local detail preservation. Mesh smoothing generalizes signal processing techniques, such as low-pass filtering, to (signals on) surfaces [BPK*08]. In this setting the fine surface details correspond to the high frequencies of the surface signal and the global shape is represented by its low frequency components. However, in contrast to surface smoothing, one now wants to explicitly modify the low frequencies and preserve the high frequency details, resulting in the desired multiresolution deformation. Fig. 11 shows a simple 2D example of this concept.

The complete multiresolution editing process is depicted in Fig. 12. In a first step 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. 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 multiresolution framework, respectively. This

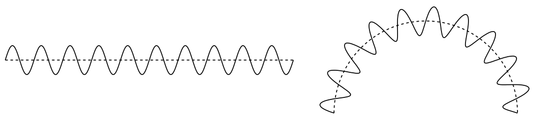


Figure 11: *A multiresolution deformation of a sine wave. A frequency decomposition yields the dashed line as its low frequency component (*left*). Bending this line and adding the higher frequencies back onto it results in the desired global shape deformation (*right*).*

multiresolution surface representation is now enhanced by an *editing* 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 multiresolution deformation $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

Notice that in general more than one decomposition step is 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 one $\mathcal{S}_{i+1}$ 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 $\mathcal{S}' = \mathcal{B}' \bigoplus_{i=0}^{k-1} D_{k-i}$. Since the generalization to several hierarchy levels is straightforward, we restrict our explanations to the simpler case of a two-band decomposition, as shown in Fig. 12.

A complete multiresolution deformation framework has to provide the three basic operators shown in Fig. 12: the decomposition operator (*detail analysis*), the editing operator (*shape deformation*), and the reconstruction operator (*detail synthesis*). The decomposition is typically performed by mesh smoothing or fairing [BPK*08], and surface deformation has been discussed above. The missing component is a suitable representation for the geometric detail $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, which we describe in the following.

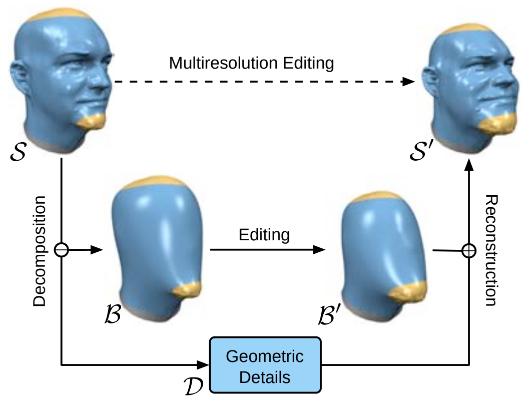


Figure 12: *A general multiresolution editing framework consists of three main operators: the* decomposition *operator, that separates the low and high frequencies, the* editing *operator, that deforms the low frequency components, and the* reconstruction *operator, that adds the details back onto the modified base surface. Since the lower part of this scheme is hidden in the multiresolution kernel, only the multiresolution edit in the top row is visible to the designer.*

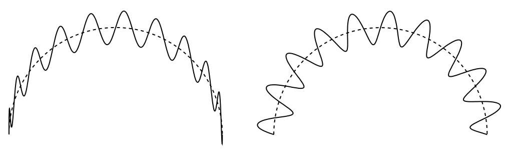


Figure 13: *Representing the displacements w.r.t. the global coordinate system does not lead to the desired result (*left*). The geometrically intuitive solution is achieved by storing the detail w.r.t. local frames that rotate according to the local tangent plane's rotation of $\mathcal{B}$ (*right*).*

4.4.1. Displacement Vectors

The straightforward representation for multiresolution details is a displacement of the base surface $\mathcal{B}$, i.e., the detail information is a vector valued displacement function $\mathbf{h} : \mathcal{B} \to \mathbb{R}^3$ that associates a displacement vector $\mathbf{h}(\mathbf{b})$ with each point $\mathbf{b}$ on the base surface. In a typical setting $\mathcal{S}$ and $\mathcal{B}$ will have the same connectivity, leading to per-vertex *displacement vectors* $\mathbf{h}_i$ [ZSS97, KCVS98, GSS99]:

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

where $\mathbf{b}_i \in \mathcal{B}$ is the vertex corresponding to $\mathbf{x}_i \in \mathcal{S}$. The vectors $\mathbf{h}_i$ have to be encoded in *local frames* w.r.t. $\mathcal{B}$ [FB88, FB95], determined by the normal vector $\mathbf{n}_i$ and two vectors spanning the tangent plane (cf. Fig. 13). 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 $\mathcal{S}'$.

4.4.2. Normal Displacements

As we will see below, long displacement vectors might lead to instabilities, in particular for bending deformations. As a consequence, for numerical robustness the displacement vectors should be as short as possible, which is the case if they connect vertices $\mathbf{x}_i \in \mathcal{S}$ to their closest surface points on $\mathcal{B}$ instead of to their corresponding vertices of $\mathcal{B}$. This idea leads to *normal displacements* that are perpendicular to $\mathcal{B}$, i.e., parallel to its normal field $\mathbf{n}$:

$$\mathbf{x}_i \;=\; \mathbf{b}_i + h_i \cdot \mathbf{n}_i \,, \quad h_i \in \mathbb{R}. \tag{13}$$

Since the displacements are in general not parallel to the surface normal, generating normal displacements has to involve some kind of resampling. Shooting rays in normal direction from each base vertex $\mathbf{b}_i \in \mathcal{B}$ and deriving new vertex positions $\mathbf{x}_i \in \mathcal{S}$ at their intersections with the detailed surface leads to a resampling of the latter [GVSS00, LMH00]. Because $\mathcal{S}$ might be a detailed surface with high frequency features, such a resampling is likely to introduce alias artifacts. Hence, Kobbelt et al. [KVS99] go the other direction: for each vertex position $\mathbf{x}_i \in \mathcal{S}$ they find a base point $\mathbf{b}_i \in \mathcal{B}$ (now *not* necessarily a vertex of $\mathcal{B}$), such that the displacements are normal to $\mathcal{B}$, i.e., $\mathbf{x}_i = \mathbf{b}_i + h_i \cdot \mathbf{n}(\mathbf{b}_i)$. This avoids a resampling of $\mathcal{S}$ and therefore allows for the preservation of all of its sharp features (see also [PKG06] 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 [BK04b].

4.4.3. Displacement Volumes

While normal displacement are extremely efficient, their main problem is that neighboring displacement vectors are not coupled in any way. When 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 (cf. Figs. 10 and 14). 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.

Both problems, the unnatural change of volume and local self-intersections, are addressed by displacement *volumes* instead of displacement *vectors* [BK03b]. Each triangle $\left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k\right)$ of $\mathcal{S}$, together with the corresponding points $\left(\mathbf{b}_i, \mathbf{b}_j, \mathbf{b}_k\right)$ on $\mathcal{B}$, defines a triangular a 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 for the original shape. The local volume preservation leads to more intuitive results and avoids local self-intersections (cf. Figs. 10, 14). However, the improved detail preservation

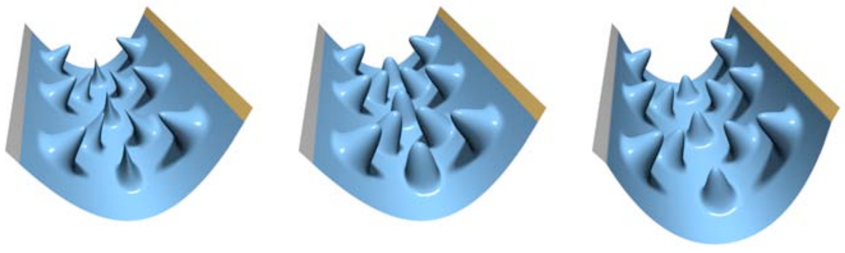


Figure 14: *For a bending of the bumpy plane, normal displacements distort geometric details and almost lead to self-intersections (*left*), whereas displacement volumes (*center*) and deformation transfer (*right*) achieve more natural results.*

comes at the higher computational cost of a nonlinear detail reconstruction process.

4.4.4. Deformation Transfer

Botsch et al. [BSPG06] use the deformation transfer approach of [SP04] to transfer the base surface deformation $\mathcal{B} \mapsto \mathcal{B}'$ onto the detailed surface $\mathcal{S}$, resulting in a multiresolution deformation $\mathcal{S}'$. This method yields results similar in quality to displacement volumes (cf. Fig. 14), but 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.

4.5. Differential Coordinates

While multiresolution or multi-scale hierarchies are an effective tool for enhancing freeform deformations by fine-scale detail preservation, the generation of the 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 will 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. More details on these topics, such as methods based on local frames [LSLC05, LCOGL07, SYBF06], sketching interfaces [NSAC05], or volumetric Laplacians [ZHS*05], can be found in the recent survey [BS07].

4.5.1. Gradient-Based Deformation

The methods of [YZX*04, ZRKS05] deform the surface by prescribing a target gradient field and finding a surface that matches this gradient field in the least squares sense. In the continuous setting, consider a function $f : \Omega \to \mathbb{R}$ that should match a user-prescribed gradient field g by minimizing

$$\int_{\Omega} \|\nabla f - g\|^2 \, \mathrm{d}u\mathrm{d}v \, .$$

Applying variational calculus yields the Euler-Lagrange equation

$$\Delta f = \mathrm{div} g \, , \tag{14}$$

which has to be solved for the optimal f.

On a discrete triangle mesh, a piecewise linear function $f : \mathcal{S} \to \mathbb{R}$ is defined by its values $f_i := f(\mathbf{x}_i)$ at the mesh vertices, which are linearly interpolated within triangles. The gradient $\nabla f : \mathcal{S} \to \mathbb{R}^3$ is a constant vector $\mathbf{g}_j \in \mathbb{R}^3$ within each triangle f_j.

If instead of a scalar function f the piecewise linear coordinate function $\mathbf{x}(v_i) = \mathbf{x}_i \in \mathbb{R}^3$ is considered, then the gradient within a face f_j is a constant 3×3 matrix

$$\nabla \mathbf{x}|_{f_j} =: \mathbf{G}_j \in \mathbb{R}^{3\times 3} \, .$$

For a mesh with V vertices and F triangles, the discrete gradient operator can be expressed by a $3F \times V$ matrix $\mathbf{G}$:

$$\begin{pmatrix} \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_F \end{pmatrix} = \mathbf{G} \cdot \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_V^T \end{pmatrix} \, .$$

The face gradients are then modified explicitly (as discussed later), yielding new gradients $\mathbf{G}'_j$ per triangle f_j. Reconstructing a mesh having these desired gradients is an overdetermined problem, and therefore is solved in a weighted least squares sense using the normal equations [GL89b]:

$$\underbrace{\mathbf{G}^T\mathbf{D}\mathbf{G}}_{\Delta_{\mathcal{S}}} \cdot \begin{pmatrix} \mathbf{x}_1'^T \\ \vdots \\ \mathbf{x}_V'^T \end{pmatrix} = \underbrace{\mathbf{G}^T\mathbf{D}}_{\mathrm{div}} \cdot \begin{pmatrix} \mathbf{G}'_1 \\ \vdots \\ \mathbf{G}'_F \end{pmatrix} \, , \tag{15}$$

where $\mathbf{D}$ is a diagonal matrix containing the face areas as weighting factors. Since the matrix $\mathbf{G}^T\mathbf{D}$ corresponds to the

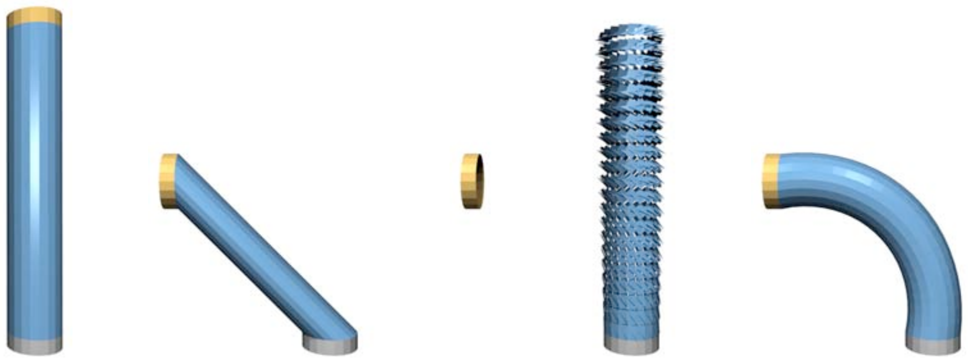


Figure 15: *Using gradient-based editing to bend the cylinder (a) by 90°. Reconstructing the mesh from new handle positions, but* original *gradients distorts the object (b). Applying damped local rotations derived from (16) to the individual triangles breaks up the mesh (c), but solving the Poisson system (15) re-connects it and yields the desired result (d).*

discrete divergence operator, and since $\operatorname{div}\nabla = \Delta$, this system actually is a Poisson equation. It corresponds to the discretization of the Euler-Lagrange PDE (14). Hence, these methods prescribe a guidance gradient field $(\mathbf{G}'_1, \ldots, \mathbf{G}'_F)$, compute its divergence, and solve three sparse linear Poisson systems for the x, y, and z coordinates of the modified mesh vertices $\mathbf{x}'_i$. An example deformation is shown in Fig. 15.

4.5.2. Laplacian-Based Deformation

Other methods manipulate Laplacians of the vertices instead of gradient fields [LSC*04, SCL*04, ZHS*05, NSAC05]. They compute initial Laplacian coordinates $\delta_i = \Delta_{\mathcal{S}}(\mathbf{x}_i)$ and manipulate them to δ'_i as discussed below. The goal is to find a new coordinate function $\mathbf{x}'$ that matches the target Laplace coordinates. In the continuous setting one has to minimize

$$\int_\Omega \left\|\Delta_{\mathcal{S}}\mathbf{x}' - \delta'\right\|^2 \mathrm{d}u\mathrm{d}v\,,$$

which leads to the Euler-Lagrange equations

$$\Delta^2_{\mathcal{S}}\mathbf{x}' = \Delta_{\mathcal{S}}\delta'\,.$$

On a discrete mesh, this yields a bi-Laplacian system to be solved for the deformed surface $\mathcal{S}'$:

$$\Delta^2_{\mathcal{S}} \cdot \begin{pmatrix} \mathbf{x}'^T_1 \\ \vdots \\ \mathbf{x}'^T_V \end{pmatrix} = \Delta_{\mathcal{S}} \cdot \begin{pmatrix} \delta'^T_1 \\ \vdots \\ \delta'^T_V \end{pmatrix}\,.$$

Although the original approaches use the uniform Laplacian discretization [LSC*04,SCL*04], the cotangent weights can be shown to yield better results for irregular triangle meshes (see Fig. 5 and [BS07]).

When we do not consider the local transformation $\delta_i \mapsto \delta'_i$, but instead reconstruct the surface from the original Laplacians δ_i, then the Euler-Lagrange equation $\Delta^2_{\mathcal{S}}\mathbf{x}' = \Delta_{\mathcal{S}}\delta$ reveals the connection to the variational bending minimization (Section 4.3), whose Euler-Lagrange PDE is $\Delta^2_{\mathcal{S}}\mathbf{x}' = 0$. Using the identities $\mathbf{x}' = \mathbf{x} + \mathbf{d}$ and $\delta = \Delta_{\mathcal{S}}\mathbf{x}$ one immediately sees that the two approaches are equivalent. The methods differ in the way they model the local rotations of geometric details or differential coordinates, either by multiresolution methods (Section 4.4) or by local transformations, as discussed in the following.

4.5.3. Local Transformations

The missing component is a technique for modifying the gradients $\mathbf{G}_j$ or Laplacians δ_i based on the affine handle transformation provided by the user. The methods discussed below derive local transformations $\mathbf{T}_i$ in order to transform gradients ($\mathbf{G}'_j = \mathbf{G}_i \cdot \mathbf{T}_j$) or Laplacians ($\delta'_i = \mathbf{T}_i \cdot \delta_i$).

The gradient-based approaches [YZX*04, ZRKS05] use the gradient of this affine deformation, i.e., its rotation and scale/shear components, for transforming the surface gradients. They first construct a smooth scalar blending field $s : \mathcal{S} \to [0,1]$ based on either geodesic distances (Section 4.2) or harmonic fields. The gradient $\mathbf{T} = \mathbf{RS}$ of the affine handle transformation $\mathbf{x} \mapsto \mathbf{Tx} + \mathbf{t}$ is decomposed into rotation $\mathbf{R}$ and scale/shear $\mathbf{S}$ using polar decomposition [SD92]. Both components are then interpolated over the support region:

$$\mathbf{T}_i = \operatorname{slerp}(\mathbf{R}, \mathbf{I}, 1 - s_i) \cdot ((1 - s_i)\mathbf{S} + s_i\mathbf{I})\,, \qquad (16)$$

where slerp$(\cdot)$ denotes quaternion interpolation, $s_i = s(\mathbf{x}_i)$ is the vertex' blending value, and $\mathbf{I}$ denotes the identity matrix. This method works well for rotations, since those are handled explicitly, but it is insensitive to handle translations: Adding a translation $\mathbf{t}$ to a given deformation does not change its gradient, and thus has no influence on the resulting surface gradients. But as there is a (nonlinear) connection between translations and local rotations of gradients, these methods yield counter-intuitive results for modifications containing large translations (Section 4.6).

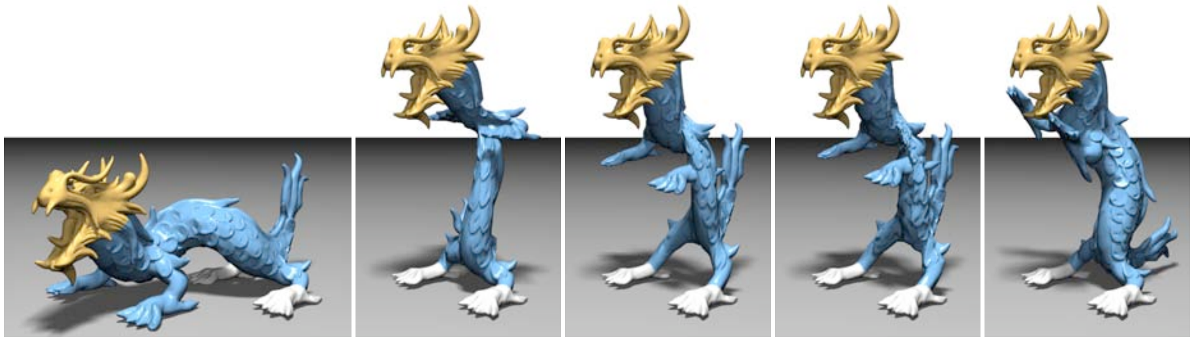


Figure 16: *The crouching dragon was lifted by fixing its hind feet and moving its head to the target position in a single, large-scale deformation. Similar to Fig. 17, the linear deformation methods yield counter-intuitive results. The nonlinear PriMo technique yields a more natural deformation.*

To address this issue, Sorkine et al. [SCL*04] implicitly optimize for the local rotations $\mathbf{T}_i$ of vertex neighborhoods by minimizing the following energy functional

$$E\left(\mathbf{x}'_1,\ldots,\mathbf{x}'_V\right) = \sum_{i=1}^{V} \left\|\mathbf{T}_i\delta_i - \Delta\left(\mathbf{x}'_i\right)\right\|^2 + \sum_{i\in\mathcal{C}} \left\|\mathbf{x}'_i - \mathbf{u}_i\right\|^2 ,$$

where $\mathbf{u}_i$ are the target positions for the constrained vertices $\mathbf{x}_i$, $i \in \mathcal{C}$. For the sake of computational efficiency they had to linearize the local frame transformations $\mathbf{T}_i$, which on the one hand allows to formulate the optimization as a single linear system, but one the other hand leads to artifacts in case of large rotations.

Lipman et al. [LSLC05, LCOGL07] minimize surface bending by preserving the relative orientations of per-vertex local frames. This is done by first solving a linear least squares system for the modified per-vertex local frame rotations $\mathbf{T}_i$, and reconstructing the modified vertex positions $\mathbf{x}'_i$ in a second step. However, since the first system does not consider the positional constraints, one has to ensure that the positional constraints and the orientation constraints are compatible. While their method works very well even for large rotations, it exhibits the same translation-insensitivity as the gradient-based methods.

4.6. Limitations of Linear Methods

In this section we compare the linear surface deformation techniques discussed so far, and point out their limitations. The goal is therefore *not* to show the best-possible results each method can produce, but rather to show under which circumstances each individual method fails. Hence, in Fig. 17 we picked extreme deformations that identify the respective limitations of the different techniques. For comparison we show the results of the non-linear surface deformation PriMo [BPGK06] (to be discussed in Section 6) , which does not suffer from linearization artifacts. For more detailed comparisons see [BS07].

The variational bending energy minimization [BK04a], in combination with the multiresolution technique [BSPG06] 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. The gradient-based editing [YZX*04, ZRKS05] updates the surface gradients using the gradient of the deformation (its rotation and scale/shear components), and therefore works very well for rotations. However, as mentioned in the last section, the explicit propagation of local rotations is translation-insensitive, such that the plane example is neither smooth nor detail preserving. The Laplacian surface editing [SCL*04] implicitly optimizes for local rotations, and hence works similarly well for translations and rotations. However, the required linearization of rotations yields artifacts for large rotations.

As the physical equations governing the surface deformation process are inherently nonlinear, all linearized techniques fail under certain circumstances. While the variational energy minimization typically works for translations, but have problems with large rotations, it is the other way around for differential approaches. Another comparison on a large-scale transformation is shown in Fig. 16. To overcome the limitations for large-scale deformations, those either have to be split up into sequences of smaller deformations — thereby complicating the user interaction — or non-linear approaches have to be considered, as discussed in Section 6.

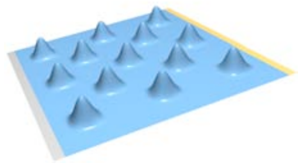
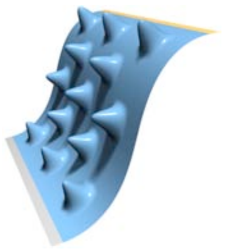
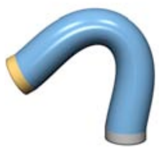
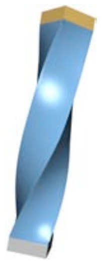
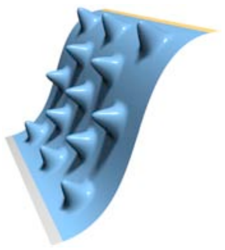
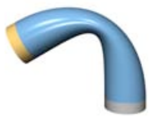
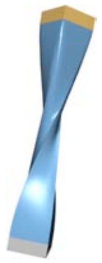
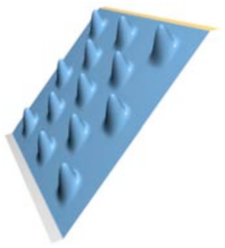
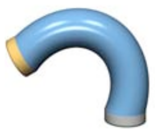
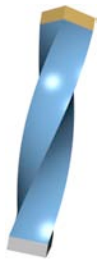
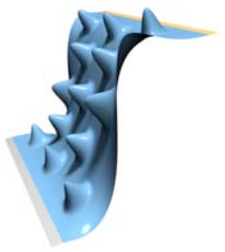
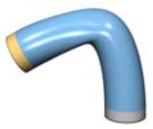
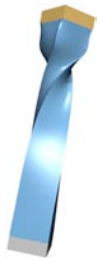
Approach	Pure Translation	120° bend	135° twist
Original model			
Nonlinear prism-based modeling [BPGK06]			
Variational minimization [BK04a] + deformation transfer [BSPG06]			
Gradient-based editing [ZRKS05]			
Laplacian-based editing with implicit optimization [SCL*04]			

Figure 17: *The extreme examples shown in this comparison matrix were particularly chosen to reveal the limitations of the respective deformation approaches.*

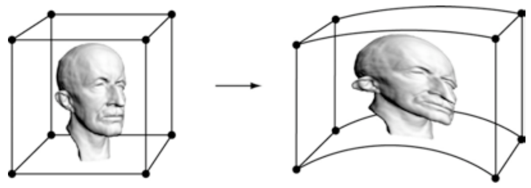


Figure 18: *Freeform space deformations warp the space around an object, and by this deform the embedded object itself.*

5. Linear Space Deformation

All the surface-based approaches described in Section 4 compute a smooth deformation field *on* the surface $\mathcal{S}$. For linear methods this typically amounts to solving a (bi-)Laplacian system as the Euler-Lagrange PDE of some quadratic energy, whereas nonlinear approaches minimize higher order energies using Newton- or Gauss-Newton-like techniques. 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 Laplacian operator is not well-defined and thus the involved linear systems become singular. Similarly, topological artifacts like gaps or non-manifold configurations lead to problems as well. In such cases quite some effort has to be spent to still be able to compute smooth deformations for the numerically problematic meshes, like eliminating degenerate triangles [BK01] or even remeshing the complete surface [BK04b]. Even when the mesh quality is sufficiently high, extremely complex meshes will result in linear or non-linear systems which cannot be solved simply due to their size.

These problems are avoided by volumetric *space deformation* techniques, that deform the ambient 3D space and by this implicitly deform the embedded objects (cf. Fig. 18). In contrast to surface-based methods, space deformation approaches employ a trivariate deformation function $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ to transform all points of the original surface $\mathcal{S}$ to the modified surface $\mathcal{S}' = \{\mathbf{p} + \mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\}$. 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.

A space deformation is defined via a (usually simple) control object; user-defined deformation of this object is interpolated to the 3D space and evaluated at the input surface points. Space deformations are typically simple to implement, and they are highly efficient and robust, because the cost of the deformation is mainly dependent on the complexity of the control object and not on the deformed shape. In the following we discuss a representative set of *linear* space deformation techniques, i.e., methods where no global nonlinear optimization is involved. These techniques are characterized by either local computations or at most global linear optimization involving the control object.

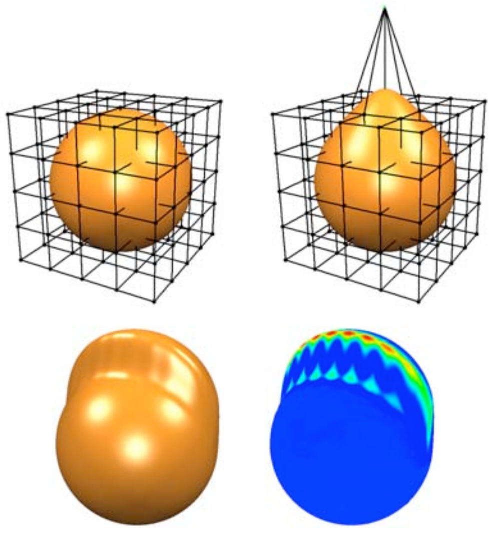


Figure 19: *In the freeform deformation approach a regular 3D control lattice is used to specify a volumetric displacement function (*left*). Similar to tensor-product spline surfaces, the tri-variate tensor-product splines can also lead to alias artifacts in the deformed surface (*right*).*

5.1. Freeform Deformation

The classical freeform deformation (FFD) method [SP86] represents the space deformation by a tensor-product Bézier or spline function

$$\mathbf{d}(u,v,w) = \sum_i \sum_j \sum_k \delta\mathbf{c}_{ijk} N_i^l(u) N_j^n(v) N_k^m(w) \quad .$$

Because of the same reasons as for spline surfaces (Section 4.1), these approaches require complex user-interactions and can cause aliasing problems, as shown in Fig. 19. In order to satisfy given displacement constraints, the inverse FFD method [HHK92] solves a linear system for the required movements of control points $\mathbf{c}_{ijk}$, which again does not necessarily imply a fair deformation of low curvature energy.

5.2. Transformation Propagation

Handle transformations can be propagated analogously to the surface-based techniques described in Section 4.2 by constructing the scalar field $s(\cdot)$ based on Euclidean distances, instead of geodesic distances [PKKG03]. While this typically leads to inferior results compared to geodesic-based propagation, this method even works if a surface-based propagation fails due to topological problems like gaps or holes.

Besides from that, the limitations of the surface-based propagation also apply to this method. A smooth interpolation of arbitrary constraints might not be possible, and the resulting surface fairness is typically inferior to techniques based on energy minimization.

5.3. Radial Basis Functions

In the case of surface-based deformations, the highest quality results are achieved by interpolating user constraints by a displacement function $\mathbf{d} : \mathcal{S} \to \mathbb{R}^3$ that additionally minimizes fairness energies (Section 4.3). Motivated by this, we therefore are looking for smoothly interpolating tri-variate *space deformation* functions $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ that minimize analogous fairness energies.

Radial basis functions (RBFs) are known to be well suited for scattered data interpolation problems [Wen05]. A trivariate RBF deformation is defined in terms of centers $\mathbf{c}_j \in \mathbb{R}^3$ and weights $\mathbf{w}_j \in \mathbb{R}^3$ as

$$\mathbf{d}(\mathbf{x}) = \sum_j \mathbf{w}_j \cdot \varphi\left(\left\|\mathbf{c}_j - \mathbf{x}\right\|\right) + \mathbf{p}(\mathbf{x}) \quad , \qquad (17)$$

where $\varphi\left(\left\|\mathbf{c}_j - \cdot\right\|\right)$ is the basis function corresponding to the jth center $\mathbf{c}_j$ and $\mathbf{p}(\mathbf{x})$ is a polynomial of low degree used to guarantee polynomial precision. In order to construct an RBF interpolating the constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$, the centers are typically placed on the constraints ($\mathbf{c}_i = \mathbf{p}_i$) and a linear system is solved for the RBF's weights $\mathbf{w}_i$ and the coefficients of the polynomial $\mathbf{p}(\mathbf{x})$ (see for instance [BK05]).

The choice of 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 be used to interpolate several hundred thousands of data points [MYC*01, OBS04], they do not provide the same degree of fairness as basis functions of global support [CBC*01]. It was shown by Duchon [Duc77] that for the basis function $\varphi(r) = r^3$ and quadratic polynomials $\mathbf{p}(\cdot) \in \Pi_2$, the function (17) is triharmonic ($\Delta^3 \mathbf{d} = 0$) and minimizes the 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 trivariate functions are *conceptually* equivalent to the minimum variation surfaces of [MS92] and the triharmonic surfaces used in [BK04a], and hence provide the same degree of fairness. The difference is that for triharmonic RBFs the energy minimization is "built-in", whereas for surface-based approaches we explicitly optimized for it (Section 4.3). The major drawback is that the fairness property comes at the price of having to solve a dense linear system, due to the global support of the triharmonic basis function $\varphi(r) = r^3$.

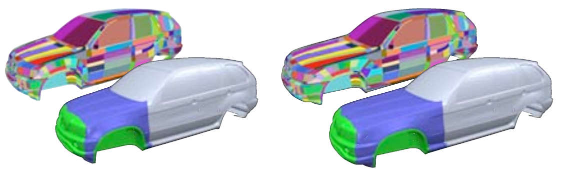


Figure 20: *Using multiple independent handle components allows to stretch the hood while rigidly preserving the shape of the wheel houses. This 3M triangle model consists of 10k individual connected components, which are neither two-manifold nor consistently oriented.*

Botsch and Kobbelt [BK05] propose an incremental least squares method that efficiently solves the linear system up to a prescribed error bound. Using this solver to pre-compute deformation basis functions allows interactively deforming even complex models. Moreover, evaluating these basis functions on the graphics card further accelerates this approach and provides real-time space deformations at a rate of 30M vertices/sec. As shown in Fig. 20, even complex surfaces consisting of disconnected patches can be handled by this technique, whereas all surface-based techniques would fail in this situation.

However, for the discussed space deformation approaches the deformed surface $\mathcal{S}'$ linearly depends on the displacement constraints $\mathbf{d}_i$. As a consequence, nonlinear effects such as local detail rotation cannot be achieved, similar to the linear surface-based methods. Although space deformations can be enhanced by multiresolution techniques as well (see, e.g., [MBK07]), they suffer from the same limitations as discussed in Section 4.6, which lead to the development of nonlinear space deformation approaches.

5.4. Cage-based techniques

As mentioned, early space deformations used lattices as control objects [SP86, Coq90]; these, however, are cumbersome to manipulate manually since the control points do not necessarily correspond to meaningful parts of the shape that the user wishes to modify. The structure of the underlying shape that is being deformed by the space warp can be easily destroyed, unless the space warp is very carefully designed.

Over the last decade it has been recognized that precise control over the properties of the deforming surface is required for more satisfactory results. Since space deformations are oblivious to the actual shape that is being edited, better control can be gained by employing control objects whose shape and structure is closely related to that of the edited shape. The *Wires* framework by Singh and Fiume [SF98] employs spatial curves to construct the deformation; the curves are aligned with prominent characteristic

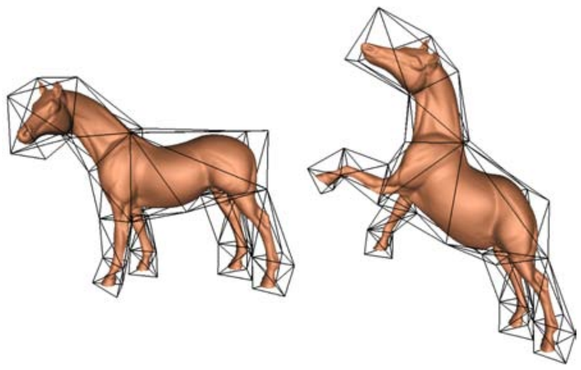


Figure 21: *Deforming a shape using cage-based space warp. The cage is a coarse representation of the shape; all points in space are represented relative to the cage using mean-value coordinates [JSW05] in this example.*

features of the edited shape and affect the surface parts in their vicinity. The framework uses a clever way to blend the space deformations induced by multiple curves.

Apart from direct surface manipulation as in the RBF space deformations above, later work proposed the use of so-called cages as control objects for shape deformations. Typically, the cage is a coarse polyhedron that can be viewed as a roughly approximating version of the input shape. The cage polyhedron is offsetted such that the edited shape resides within its interior.

Let us denote the set of cage vertices by $\{\mathbf{v}_i\}_{i=1}^n$ and the set of triangles by $\{t_j\}_{j=1}^k$. The basic approach of cage-based methods is to define coordinate functions, dependent on the vertices (and possibly the faces) of the cage polyhedron, such that each point $\mathbf{p}$ in the interior of the cage is expressed as a linear combination of the cage's elements. The mean-value coordinates [Flo03, JSW05] and the harmonic coordinates [JMD*07] express points in space as affine combinations of the vertices of the cage:

$$\mathbf{p} = \sum_{i=1}^{n} \phi_i(\mathbf{p})\mathbf{v}_i\,.$$

When the cage object is manipulated and its vertices are displaced to new positions $\mathbf{v}'_i$, the induced space deformation $\mathbf{p} \to \mathbf{p}'$ is simply

$$\mathbf{p}' = \sum_{i=1}^{n} \phi_i(\mathbf{p})\mathbf{v}'_i\,.$$

The coordinate functions $\phi_i(\cdot)$ are smooth, and in the case of mean-value coordinates and harmonic coordinates they are affine-invariant (hence, if the cage undergoes an affine transformation, the coordinates reproduce that transformation). The mean-value coordinates have a relatively simple closed formulation [JSW05]; however, they can be negative, which may lead to unintuitive deformations. This problem is partially addressed in [LKCOL07], making the mean-value coordinates non-negative at the expense of their smoothness. The harmonic coordinates [JMD*07] are smooth, non-negative functions that do not have extrema in the interior of the cage. They are obtained by solving the Laplace equation $\Delta\phi_i = 0$ with boundary conditions $\phi_i(\mathbf{v}_j) = \delta_{i,j}$. The harmonic coordinates provide better deformation results, at the expense of loosing the closed-form formulation and having to solve a linear PDE on the cage volume.

Since affine transformations include shearing, affine-invariant coordinates easily admit distorting shearing artifacts. The Green coordinates [LLCO08] allow shape-preserving deformations by removing the affine invariance property. These coordinates are defined in terms of the vertices *and face normals* of the cage:

$$\mathbf{p} = \sum_{i=1}^{n} \phi_i(\mathbf{p})\mathbf{v}_i + \sum_{j=1}^{k} \psi_j(\mathbf{p})\mathbf{n}(t_j)\,,$$

where $\mathbf{n}(t_j)$ is the unit normal of face t_j. Using Green's functions and Green's integral identities, it is possible to show closed formulas for the Green coordinate functions in 2D and 3D, such that the resulting deformations are (quasi-)conformal.

6. Nonlinear Surface Deformation

Thanks to the rapid increase in both computational power and available memory of today's workstations, nonlinear deformation methods become more and more tractable, which in the last years already lead to a first set of nonlinear, yet interactive, surface deformation approaches. While a nonlinear implementation of the previously discussed approaches may seem straightforward (simply do not use any linearization), special attention must be paid to computational efficiency and numerical robustness.

As discussed in Section 4.5, the nonlinearity of surface deformation stems from the fact that local translation and rotation transformations need to be coupled in a nonlinear manner to yield intuitive results. While linear approaches either linearize the relationship between translations and rotations [SCL*04] or decouple it altogether [YZX*04, LSLC05], admitting nonlinear formulation allows to attack the problem more directly.

One of the first attempts at nonlinear surface-based deformation was the pyramid coordinate approach by Sheffer and Kraevoy [SK04, KS06]. Pyramid coordinates can be considered as a nonlinear version of the Laplacian coordinates: these are differential coordinates invariant under rigid motions. The coordinates encode the relationship between a vertex and its 1-ring by representing the vertex as a sum of the normal component and the tangential component, the latter being encoded using mean value coordinates [Flo03]. The pyramid coordinates can be effectively used for translation-sensitive mesh editing and mesh morphing; however, the nonlinear optimization involved is quite complex and requires a multi-resolution mesh hierarchy [KS06].

The use of nonlinear differential coordinates persisted in several works, such as [HSL*06,SZT*07]. The optimization problem

$$\min_{\mathbf{p}'} \|\Delta_{\mathcal{S}}\mathbf{p}' - \delta\|^2$$

becomes nonlinear when one replaces the Laplace operator of the reference surfaces, $\Delta_{\mathcal{S}}$, by the Laplacian of the target surface, $\Delta_{\mathcal{S}'}$, as was done in those works. In particular, Huang et al. [HSL*06] employ a nonlinear version of the volumetric graph Laplacian, which also features nonlinear volume preservation constraints. In order to increase performance and efficiency of their optimization they use a subspace approach: The original mesh is embedded in a coarse control mesh, i.e., a cage, and the optimization is performed on the control mesh. The vertices of the original mesh are encoded w.r.t. the cage using mean-value coordinates for polyhedra (as defined in [JSW05], see also Section 5); the constraints posed by the user on the original high-resolution mesh can then be expressed in terms of the cage mesh in a least-squares manner. Shi et al. [SZT*07] employ a skeleton as a subspace, instead of a cage. This allows for easy and intuitive character posing, because specific constraints can be formulated for inverse kinematics, rigidity and length preservation of limbs, joint angle limits and balance under gravity. The resulting nonlinear optimization is quite complex, however: careful weight selection for the various energy terms is required, as well as an intricate cascading optimization of the total energy.

An alternative approach to subspace methods is the handle-aware isoline technique of [AFTCO07]. In a preprocessing step they construct a set of isolines of the geodesic distance from either the fixed regions or the handle regions, similar in spirit to [ZRKS05]. For each of these isolines they find a local transformation $\mathbf{T}_i$ for a Laplacian-based deformation, based on a nonlinear optimization. The number of required isolines is relatively small, which guarantees an efficient numerical optimization and thereby allows for interactive editing.

The latest nonlinear approaches use variational minimization of the bending and stretching energies. The PRIMO system [BPGK06] defines a mesh deformation energy, coupled with a specially tailored minimization technique, such that large deformations can be performed robustly. In the rest state, each mesh face is associated with a prism, obtained by extruding each mesh vertex along the normal direction by a fixed distance. The prisms are thought of as *rigid* entities; in the rest state adjacent prisms share faces, but as the surface is deformed, they break apart, creating an imaginary volume enclosed between the corresponding faces of adjacent prisms. The PRIMO energy is the integral of the infinitesimal elastic forces between each pair of adjacent prism faces, which is proportional to the size of that enclosed volume. Minimizing the energy amounts to finding an optimal set of rigid transformations (rotations and translations) for all the prisms such that the elastic forces are minimized. This is done in a hierarchical manner, where the hierarchy levels are constructed by successive clustering of neighboring prisms into one rigid group. On the coarsest hierarchy level, the optimal rigid transformations are found by a global non-linear optimization (using Newton iterations with line search); on finer levels the coarse transformations are improved by iterative local shape matching: in each iteration, all prisms are kept fixed except for a randomly chosen one, for which the optimal rigid transformation can then be computed in closed form by solving a 4×4 eigenproblem [Hor87]. The surface deformation technique is robust thanks to the fact that the prisms are kept rigid and are not allowed to degenerate; this constrains the optimization and prevents running into locally minimal configurations that allow surface folds and other degeneracies.

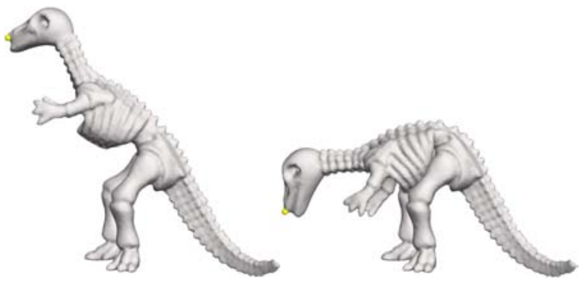


Figure 22: *Nonlinear surface-based deformation allows for large bending while preserving the surface detail (example from [SA07]).*

Similar in spirit, the as-rigid-as-possible surface deformation of [SA07] models local rotations in terms of each vertex's 1-ring. The optimal local rotation $\mathbf{R}_i$ is defined as the one that minimizes

$$\sum_{v_j \in \mathcal{N}_1(v_i)} \| (\mathbf{p}'_i - \mathbf{p}'_j) - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \|^2.$$

$\mathbf{R}_i$ can be computed by the method of [Hor87] and is a nonlinear function of $\mathbf{p}'$. The total deformation energy

$$\sum_{v_i} \sum_{v_j \in \mathcal{N}_1(v_i)} w_{ij} \| (\mathbf{p}'_i - \mathbf{p}'_j) - \mathbf{R}_i (\mathbf{p}_i - \mathbf{p}_j) \|^2.$$

is minimized by alternating optimization: fixing $\mathbf{p}'$ allows to solve for $\mathbf{R}_i$ (independently for each vertex), and fixing $\mathbf{R}_i$ allows to update the positions $\mathbf{p}'$ by solving a linear Laplacian-type system. The weights w_{ij} are the cotangent weights defined in Eq. (7). The overall optimization is efficient and relies on simple linear components; the alternating iterations are guaranteed to decrease the energy in each step.

An extension of nonlinear gradient-based deformation to mesh sequences was presented by [XZY*07]. They employed a similar alternating optimization approach, while incorporating time-coherence constraints into the system.

7. Nonlinear Space Deformation

Here we briefly review advanced space deformation techniques which typically involve nonlinear optimization.

Schaefer and colleagues [SMW06] proposed a moving-least-squares approach to space deformations, in which the transformation at each point $\mathbf{x}$ in space is defined by a weighted least-squares optimization:

$$\mathbf{T}(\mathbf{x}) = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i w_i(\mathbf{x}) \|T(\mathbf{p}_i) - \mathbf{u}_i\|^2,$$

where $\mathbf{p}_i$ are some points in space which the user wishes to transform to positions $\mathbf{u}_i$. The weights $w_i(\mathbf{x})$ depend on the point where the moving-least-squares function is evaluated, e.g.,

$$w_i(\mathbf{x}) = \frac{1}{\|\mathbf{p}_i - \mathbf{x}\|^{2\alpha}}.$$

The transformations $\mathbf{T}$ should be restricted to a particular class, such as similarity or rotation; in 2D it can be then shown that the optimization problem is linear, whereas in 3D rotations cannot be linearly parameterized, and an eigenproblem solution is required [ZG07].

Sumner et al. [SSP07] 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. In addition to static geometries, their method can also be applied to handcrafted animations and precomputed simulations.

Botsch et al. [BPWG07] extend the PriMo framework [BPGK06] to deformations of solid objects. 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 [ACWK06, vFTS06]. The advantage of those techniques is that they by construction yield volume preserving and intersection-free deformations. As a drawback, it is harder to construct vector fields that exactly satisfy user-defined deformation constraints.

8. Conclusion

Shape deformation continues to be an interesting and challenging area of research. As computer power increases, it becomes possible to conceive more and more involved optimization targets, tailored to the particular needs of the application at hand. We anticipate further development of surface deformation techniques, especially nonlinear methods, as well as further work on improving the robustness and efficiency of those methods, leading to eventual integration into the modeling software used in the industry.

Acknowledgments

We are grateful to Mark Pauly, Leif Kobbelt, Pierre Alliez, and Bruno Lévy. Parts of our course notes are based on the notes of a more general course on Geometric Modeling with Polygonal Meshes [BPK*08] co-authored with them.

Appendix A: Numerics

In this section we describe different types of solvers for sparse linear systems. Within this class of systems, we will further concentrate on symmetric positive definite (so-called *spd*) matrices, since exploiting their special structure allows for the most efficient and most robust implementations. Examples of such matrices are Laplacian system (to be analyzed in Section A) and general least squares systems.

Following [BBK05], we propose the use of direct solvers for sparse spd systems, since their superior efficiency — although well known in the field of high performance computing — is often neglected in geometry processing applications. After reviewing the commonly known and used direct and iterative solvers, we introduce sparse direct solvers and point out their advantages.

For the following discussion we restrict ourselves to sparse spd problems $\mathbf{Ax} = \mathbf{b}$, with $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n \times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, and denote by $\mathbf{x}^*$ the exact solution $\mathbf{A}^{-1}\mathbf{b}$.

Laplacian Systems

Since Laplacian systems played a major role in the deformation approaches discussed in this paper (Sections 4.3, 4.5), we will shortly describe general Laplacian matrices first.

In each row the matrix $\Delta_\mathcal{S}$ contains the weights for the discretization of the Laplace-Beltrami of a function $f : \mathcal{S} \to \mathbb{R}$ at one vertex v_i (see Chapter 3):

$$\Delta_\mathcal{S} f(v_i) = \frac{2}{A(v_i)} \sum_{v_j \in \mathcal{N}_1(v_i)} \left(\cot\alpha_{ij} + \cot\beta_{ij}\right) \left(f(v_j) - f(v_i)\right).$$

This can be written in matrix notation as

$$\begin{pmatrix} \vdots \\ \Delta_\mathcal{S} f(v_i) \\ \vdots \end{pmatrix} = \mathbf{D} \cdot \mathbf{M} \cdot \begin{pmatrix} \vdots \\ f(v_i) \\ \vdots \end{pmatrix},$$

where $\mathbf{D}$ is a diagonal matrix of normalization factors $\mathbf{D}_{ii} = 2/A(v_i)$, and $\mathbf{M}$ is a symmetric matrix containing the cotangent weights. Since the Laplacian of a vertex v_i is defined *locally* in terms of its one-ring neighbors, the matrix $\mathbf{M}$ is highly sparse and has non-zeros in the ith row only on the diagonal and in those columns corresponding to v_i's one-ring neighbors $\mathcal{N}_1(v_i)$.

For a closed mesh, Laplacian systems $\Delta_S^k \mathbf{P} = \mathbf{B}$ of any order k can be turned into symmetric ones by moving the first diagonal matrix $\mathbf{D}$ to the right-hand side:

$$\mathbf{M}(\mathbf{DM})^{k-1}\mathbf{P} = \mathbf{D}^{-1}\mathbf{B} \ . \qquad (18)$$

Boundary constraints are typically employed by restricting the values at certain vertices, which corresponds to eliminating their respective rows and columns and hence keeps the matrix symmetric. The case of meshes with boundaries is equivalent to a patch bounded by constrained vertices and therefore also results in a symmetric matrix. Pinkal and Polthier [PP93] additionally showed that this system is positive definite, such that the efficient solvers presented in the next section can be applied.

Dense Direct Solvers

Direct linear system solvers are based on a factorization of the matrix $\mathbf{A}$ into matrices of simpler structure, e.g., triangular, diagonal, or orthogonal matrices. This structure allows for an efficient solution of the factorized system. As a consequence, once the factorization is computed, it can be used to solve the linear system for several different right hand sides.

The most commonly used examples for *general* matrices $\mathbf{A}$ are, in the order of increasing numerical robustness and computational effort, the LU factorization, QR factorization, or the singular value decomposition. However, in the special case of a spd matrix the Cholesky factorization $\mathbf{A} = \mathbf{LL}^T$, with $\mathbf{L}$ denoting a lower triangular matrix, should be employed, since it exploits the symmetry of the matrix and can additionally be shown to be numerically very robust due to the positive definiteness of the matrix $\mathbf{A}$ [GL89b].

On the downside, the asymptotic time complexity of all dense direct methods is $O(n^3)$ for the factorization and $O(n^2)$ for solving the system based on the pre-computed factorization. Since for the problems we are targeting at, n can be of the order of 10^5, the total cubic complexity of dense direct methods is prohibitive. Even if the matrix $\mathbf{A}$ is highly sparse, the naïve direct methods enumerated here are not designed to exploit this structure, hence the factors are dense matrices in general (cf. Fig. 24, top row).

Iterative Solvers

In contrast to dense direct solvers, iterative methods are able to exploit the sparsity of the matrix $\mathbf{A}$. Since they additionally allow for a simple implementation [PFTV92], iterative solvers are the de-facto standard method for solving sparse linear systems in the context of geometric problems. A detailed overview of iterative methods with valuable implementation hints can be found in [BBC*94].

Iterative methods compute a converging sequence $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i)}$ of approximations to the solution $\mathbf{x}^*$ of the linear system, i.e., $\lim_{i\to\infty} \mathbf{x}^{(i)} = \mathbf{x}^*$. In practice, however, one has to find a suitable criterion to stop the iteration if the current solution $\mathbf{x}^{(i)}$ is accurate enough, i.e., if the norm of the error $\mathbf{e}^{(i)} := \mathbf{x}^* - \mathbf{x}^{(i)}$ is less than some ε. Since the solution $\mathbf{x}^*$ is not known beforehand, the error has to be estimated by considering the residual $\mathbf{r}^{(i)} := \mathbf{A}\mathbf{x}^{(i)} - \mathbf{b}$. These two are related by the *residual equations* $\mathbf{A}\mathbf{e}^{(i)} = \mathbf{r}^{(i)}$, leading to an upper bound $\left\|\mathbf{e}^{(i)}\right\| \leq \left\|\mathbf{A}^{-1}\right\| \cdot \left\|\mathbf{r}^{(i)}\right\|$, i.e., the norm of the inverse matrix has to be estimated or approximated in some way (see [BBC*94]).

In the case of spd matrices the method of conjugate gradients (CG) [GL89b, She94] is suited best, since it provides guaranteed convergence with monotonically decreasing error. For a spd matrix $\mathbf{A}$ the solution of $\mathbf{Ax} = \mathbf{b}$ is equivalent to the minimization of the quadratic form

$$\phi(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \ .$$

The CG method successively minimizes this functional along a set of linearly independent $\mathbf{A}$-*conjugate* search directions, such that the exact solution $\mathbf{x}^* \in \mathbb{R}^n$ is found after at most n steps (neglecting rounding errors). The complexity of each CG iteration is mainly determined by the matrix-vector product $\mathbf{Ax}$, which is of order $O(n)$ if the matrix is sparse. Given the maximum number of n iterations, the total complexity is $O(n^2)$ in the worst case, but it is usually better in practice.

As the convergence rate mainly depends on the spectral properties of the matrix $\mathbf{A}$, a proper pre-conditioning scheme should be used to increase the efficiency and robustness of the iterative scheme. This means that a slightly different system $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ is solved instead, with $\tilde{\mathbf{A}} = \mathbf{PAP}^T$, $\tilde{\mathbf{x}} = \mathbf{P}^{-T}\mathbf{x}$, $\tilde{\mathbf{b}} = \mathbf{Pb}$, using a regular pre-conditioning matrix $\mathbf{P}$, that is chosen such that $\tilde{\mathbf{A}}$ is well conditioned [GL89b, BBC*94]. However, the matrix $\mathbf{P}$ is restricted to have a simple structure, since an additional linear system $\mathbf{Pz} = \mathbf{r}$ has to be solved each iteration.

The iterative conjugate gradients method manages to decrease the computational complexity from $O(n^3)$ to $O(n^2)$ for sparse matrices. However, this is still too slow to compute exact (or sufficiently accurate) solutions of large and possibly ill-conditioned systems.

Multigrid Iterative Solvers

One characteristic problem of most iterative solvers is that they are *smoothers*: they attenuate the high frequencies of the error $\mathbf{e}^{(i)}$ very fast, but their convergence stalls if the error is a smooth function. This fact is exploited by multigrid methods, that build a fine-to-coarse hierarchy $\{\mathcal{M} = \mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k\}$ of the computation domain $\mathcal{M}$

and solve the linear system hierarchically from coarse to fine [Hac86, BHM00].

After a few (pre-)smoothing iterations on the finest level $\mathcal{M}_0$ the high frequencies of the error are removed and the solver becomes inefficient. However, the remaining low frequency error $\mathbf{e}_0 = \mathbf{x}^* - \mathbf{x}_0$ on $\mathcal{M}_0$ corresponds to higher frequencies when restricted to the coarser level $\mathcal{M}_1$ and therefore can be removed efficiently on $\mathcal{M}_1$. Hence the error is solved for using the residual equations $\mathbf{A}\mathbf{e}_1 = \mathbf{r}_1$ on $\mathcal{M}_1$, where $\mathbf{r}_1 = R_{0\to1}\mathbf{r}_0$ is the residual on $\mathcal{M}_0$ transferred to $\mathcal{M}_1$ by a restriction operator $R_{0\to1}$. The result is prolongated back to $\mathcal{M}_0$ by $\mathbf{e}_0 \leftarrow P_{1\to0}\mathbf{e}_1$ and used to correct the current approximation: $\mathbf{x}_0 \leftarrow \mathbf{x}_0 + \mathbf{e}_0$. Small high-frequency errors due to the prolongation are finally removed by a few post-smoothing steps on $\mathcal{M}_0$. The recursive application of this two-level approach to the whole hierarchy can be written as

$$\Phi_i \;=\; S_\mu P_{i+1\to i}\,\Phi_{i+1}\,R_{i\to i+1}\,S_\lambda \quad ,$$

with λ and μ pre- and post-smoothing iterations, respectively. One recursive run is known as a *V-cycle* iteration.

Another concept is the method of *nested iterations*, that exploits the fact that iterative solvers are very efficient if the starting value is sufficiently close to the actual solution. One starts by computing the exact solution on the coarsest level $\mathcal{M}_k$, which can be done efficiently since the system $\mathbf{A}_k\mathbf{x}_k = \mathbf{b}_k$ corresponding to the restriction to $\mathcal{M}_k$ is small. The prolongated solution $P_{k\to k-1}\mathbf{x}_k^*$ is then used as starting value for iterations on $\mathcal{M}_{k-1}$, and this process is repeated until the finest level $\mathcal{M}_0$ is reached and the solution $\mathbf{x}_0^* = \mathbf{x}^*$ is computed.

The remaining question is how to iteratively solve on each level. The standard method is to use one or two V-cycle iterations, leading to the so-called *full multigrid* method. However, one can also use an iterative smoothing solver (e.g., Jacobi or CG) on each level and completely avoid V-cycles. In the latter case the number of iterations m_i on level i must not be constant, but instead has to be chosen as $m_i = m\gamma^i$ to decrease exponentially from coarse to fine [BD96]. Besides the easier implementation, the advantage of this *cascading multigrid* method is that once a level is computed, it is not involved in further computations and can be discarded. A comparison of the three methods in terms of visited multigrid levels is given in Fig. 23.

Due to the logarithmic number of hierarchy levels $k = O(\log n)$ the full multigrid method and the cascading multigrid method can both be shown to have linear asymptotic complexity, as opposed to quadratic for non-hierarchical iterative methods. However, they cannot exploit synergy for multiple right hand sides, which is why factorization-based approaches are clearly preferable in such situations, as we will show in the next section.

Since in our case the discrete computational domain $\mathcal{M}$ is an irregular triangle mesh instead of a regular 2D or 3D grid,

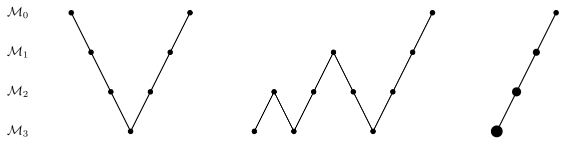


Figure 23: *A schematic comparison in terms of visited multigrid levels for V-cycle* (left)*, full multigrid with one V-cycle per level* (center)*, and cascading multigrid* (right)*.*

the coarsening operator for building the hierarchy is based on mesh decimation techniques [KCS98]. The shape of the resulting triangles is important for numerical robustness, and the edge lengths on the different levels should mimic the case of regular grids. Therefore the decimation usually removes edges in the order of increasing lengths, such that the hierarchy levels have uniform edge lengths and triangles of bounded aspect ratio. The simplification from one hierarchy level $\mathcal{M}_i$ to the next coarser one $\mathcal{M}_{i+1}$ should additionally be restricted to remove a *maximally independent set* of vertices, i.e., no two removed vertices $v_j, v_l \in \mathcal{M}_i \setminus \mathcal{M}_{i+1}$ are connected by an edge $e_{jl} \in \mathcal{M}_i$. In [AKS05] some more efficient alternatives to this kind of hierarchy are described.

The linear complexity of multi-grid methods allows for the highly efficient solution even of very complex systems. However, the main problem of these solvers is their quite involved implementation, since special care has to be taken for the hierarchy building, for special multigrid preconditioners, and for the inter-level conversion by restriction and prolongation operators. Additionally, appropriate numbers of iterations per hierarchy level have to be chosen. These numbers have to be chosen either by heuristic or experience, since they not only depend on the problem (structure of $\mathbf{A}$), but also on its specific instance (values of $\mathbf{A}$). A detailed overview of these techniques is given in [AKS05]. A highly efficient multigrid solver with specially tuned restriction and prolongation operators was proposed for interactive shape deformation in [SYBF06].

Sparse Direct Solvers

The use of direct solvers for large sparse linear systems is often neglected, since naïve direct methods have complexity $O(n^3)$, as described above. The problem is that even when the matrix $\mathbf{A}$ is sparse, the factorization will not preserve this sparsity, such that the resulting Cholesky factor is a dense lower triangular matrix.

However, an analysis of the factorization process reveals that a *band-limitation* of the matrix $\mathbf{A}$ will be preserved. If the matrix $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ has a certain bandwidth β then so has its factor $\mathbf{L}$. An even stricter bound is that the so-called *envelope* (the leading zeros of each row) is preserved [GL81]. This additional structure can be exploited in both the factorization and the solution process, such that their complexities reduce from $O(n^3)$ and $O(n^2)$ to linear complexity in the number of non-zeros $\mathrm{nz}(\mathbf{A})$ of $\mathbf{A}$ [GL81]. Since usually $\mathrm{nz}(\mathbf{A}) = O(n)$, this is the same linear complexity as for multigrid solvers. However, in particular for multiple right-hand side problems, sparse direct methods turned out to be more efficient compared to multigrid solvers.

If matrices are sparse, but not band-limited or profile-optimized, the first step is to minimize the matrix envelope, which can be achieved by symmetric row and column permutations $\mathbf{A} \leftarrow \mathbf{P}^T\mathbf{A}\mathbf{P}$ using a permutation matrix $\mathbf{P}$, i.e., a re-ordering of the mesh vertices. Although this problem is NP complete, several good heuristics exist, of which we will outline the most commonly used in the following. All of these methods work on the undirected *adjacency graph* $\mathrm{Adj}(\mathbf{A})$ corresponding to the non-zeros of $\mathbf{A}$, i.e., two nodes $i, j \in \{1, \ldots, n\}$ are connected by an edge if and only if $\mathbf{A}_{ij} \neq 0$.

The standard method for envelope minimization is the *Cuthill-McKee* algorithm [CM69], that picks a start node and renumbers all its neighbors by traversing the adjacency graph in a greedy breadth-first manner. Reverting this permutation further improves the re-ordering, leading to the *reverse Cuthill-McKee* method (RCMK) [LS76]. The result $\mathbf{P}^T\mathbf{A}\mathbf{P}$ of this matrix re-ordering is depicted in the second row of Fig. 24.

Since no special pivoting is required for the Cholesky factorization, the non-zero structure of its matrix factor $\mathbf{L}$ can symbolically be derived from the non-zero structure of the matrix $\mathbf{A}$ alone, or, equivalently, from its adjacency graph. The *minimum degree* algorithm (MD) and its variants [GL89a, Liu85] directly work on the graph interpretation of the Cholesky factorization and try to minimize fill-in elements $\mathbf{L}_{ij} \neq 0 = \mathbf{A}_{ij}$. While the resulting re-orderings do not yield a band-structure (which implicitly limits fill-in), they usually lead to better results compared to RCMK (cf. Fig. 24, third row).

The last class of re-ordering approaches is based on graph partitioning. A matrix $\mathbf{A}$ whose adjacency graph has m separate connected components can be restructured to a block-diagonal matrix of m blocks, such that the factorization can be performed on each block individually. If the adjacency graph is connected, a small subset S of nodes, whose elimination would separate the graph into two components of roughly equal size, is found by one of several heuristics [KK98]. This graph-partitioning results in a matrix consisting of two large diagonal blocks (two connected components) and $|S|$ rows representing their connection (separator S). Recursively repeating this process leads to the method of *nested dissection* (ND), resulting in matrices of the typical block structure shown in the bottom row of Fig. 24. Besides the obvious fill-in reduction, these systems also allow for easy parallelization of both the factorization and the solution.

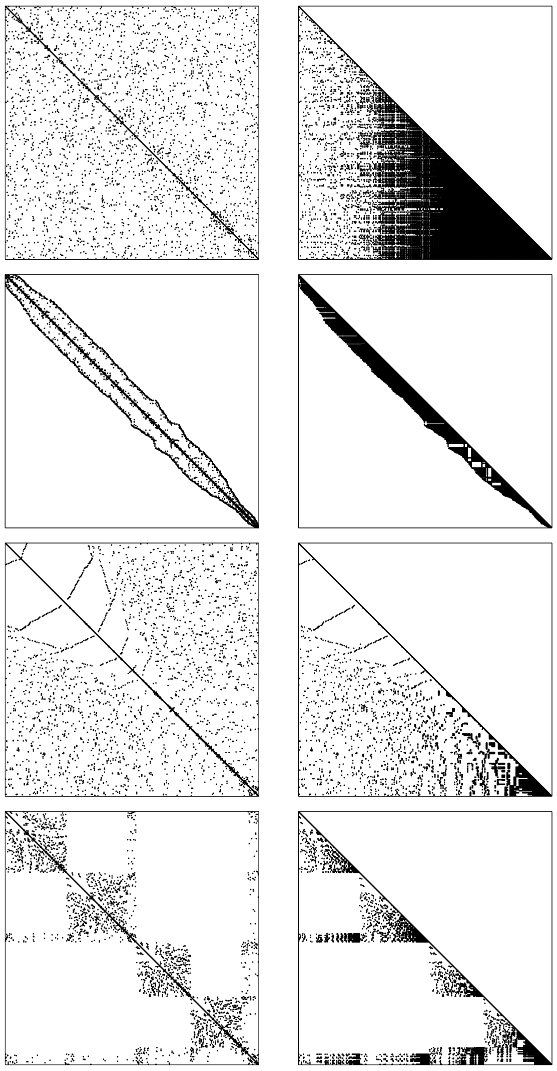


Figure 24: *The top row shows the non-zero pattern of a typical* 500×500 *matrix* $\mathbf{A}$ *and its Cholesky factor* $\mathbf{L}$*, corresponding to a Laplacian system on a triangle mesh. Although* $\mathbf{A}$ *is highly sparse (3502 non-zeros), the factor* $\mathbf{L}$ *is dense (36k non-zeros). The reverse Cuthill-McKee algorithm minimizes the envelope of the matrix, resulting in 14k non-zeros of* $\mathbf{L}$ (2nd row)*. The minimum degree ordering avoids fill-in during the factorization, which decreases the number of non-zeros to 6203* (3rd row)*. The last row shows the result of a nested dissection method (7142 non-zeros), that allows for parallelization due to its block structure.*

Analogously to the dense direct solvers, the factorization can be exploited to solve for different right hand sides in a very efficient manner, since only the back-substitution has to be performed again. Moreover, for sparse direct methods no additional parameters have to be chosen in a problem-dependent manner (like iteration numbers for iterative solvers). The only degree of freedom is the matrix re-ordering, which only depends on the symbolic structure of the problem and therefore can be chosen quite easily. A highly efficient implementation is publicly available in the TAUCS library [TCR03] or recently in COLMOD [DH05].

Comparison

In the following we compare the different kinds of linear system solvers for Laplacian as well as for bi-Laplacian systems. All timings reported in this and the next section were taken on a 3.0GHz Pentium4 running Linux. The iterative solver (*CG*) from the `gmm++` library [RP05] is based on the conjugate gradients method and uses an incomplete $\mathbf{LDL}^T$ factorization as preconditioner. The cascading multigrid solver of [BK04a] (*MG*) performs preconditioned conjugate gradient iterations on each hierarchy level and additionally exploits SSE instructions in order to solve for up to four right-hand sides simultaneously. The direct solver (LL^T) of the TAUCS library [TCR03] employs nested dissection re-ordering and a sparse complete Cholesky factorization. Although our linear systems are symmetric, we also compare to the popular SuperLU solver [DEG*99], which is based on a sparse LU factorization.

Iterative solvers have the advantage over direct ones that the computation can be stopped as soon as a sufficiently small error is reached, which — in typical computer graphics applications — does not have to be the highest possible precision. In contrast, direct methods always compute the exact solution up to numerical round-off errors, which in our application examples actually was more precise than required. The stopping criteria of the iterative methods have therefore been chosen to yield sufficient results, such that their quality is comparable to that achieved by direct solvers. The resulting residual errors were allowed to be about one order of magnitude larger than those of the direct solvers.

Table 1 shows timings for the different solvers on Laplacian systems $\Delta_{\mathcal{S}}\mathbf{P} = \mathbf{B}$ of 10k to 50k and 100k to 500k unknowns. For each solver three columns of timings are given:

Setup: Computing the cotangent weights for the Laplace discretization and building the matrix structure (done per-level for the multigrid solver).

Precomputation: Preconditioning (*iterative*), building the hierarchy by mesh decimation (*multigrid*), matrix re-ordering and sparse factorization (*direct*).

Solution: Solving the linear system for three different right-hand sides corresponding to the x, y, and z components of the free vertices $\mathbf{P}$.

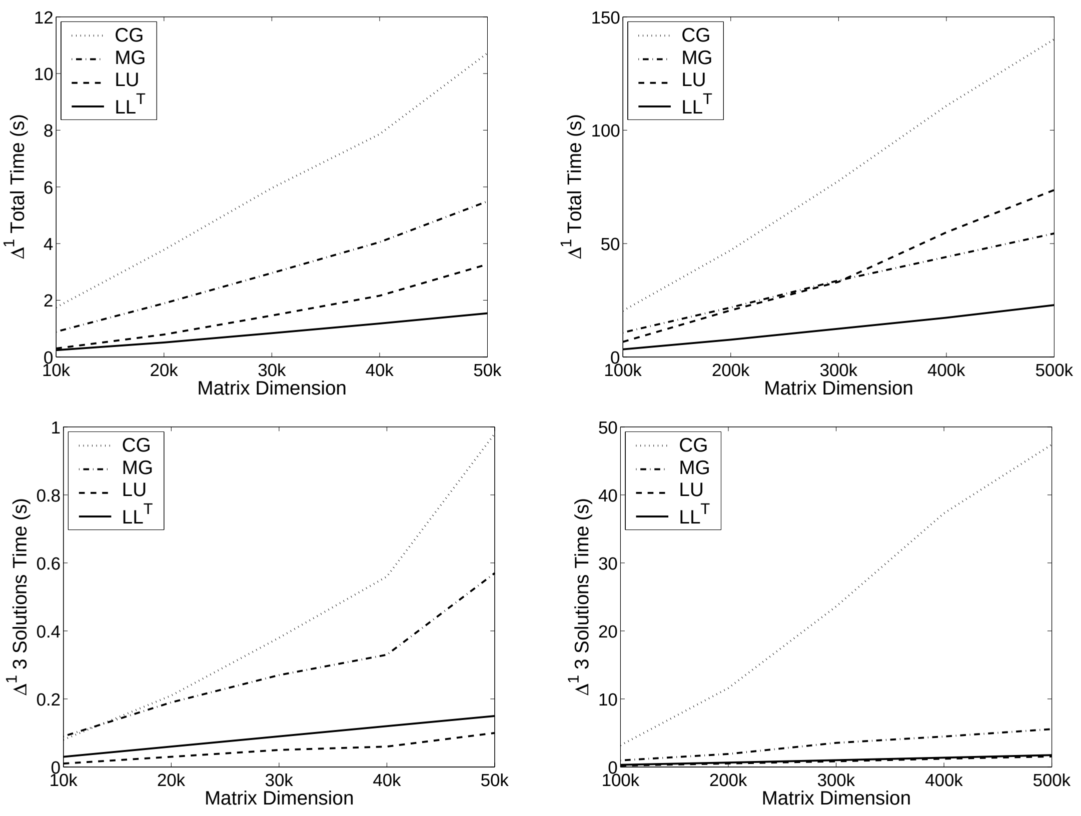
Due to its effective preconditioner, which computes a sparse incomplete factorization, the iterative solver scales almost linearly with the system complexity. However, for large and thus ill-conditioned systems it breaks down. Notice that without preconditioning the solver would not converge for the larger systems. The experiments clearly verify the linear complexity of multigrid and sparse direct solvers. Once their sparse factorizations are pre-computed, the computational costs for actually solving the system are about the same for the LU and Cholesky solver. However, they differ significantly in the factorization performance, because the numerically more robust Cholesky factorization allows for more optimizations, whereas pivoting is required for the LU factorization to guarantee robustness. This is the reason for the break-down of the LU solver, such that the multigrid solver is more efficient in terms of total computation time for the larger systems.

Interactive applications often require to solve the same linear system for several right-hand sides (e.g. once per frame), which typically reflects the change of boundary constraints due to user interaction. For such problems the solution times, i.e., the third columns of the timings, are more relevant, as they correspond to the per-frame computational costs. Here the precomputation of a sparse factorization pays off and the direct solvers are clearly superior to the multigrid method.

Table 2 shows the same experiments for bi-Laplacian systems $\Delta_{\mathcal{S}}^2\mathbf{P} = \mathbf{B}$ of the same complexity. In this case, the matrix setup is more complex, the matrix condition number is squared, and the sparsity decreases from 7 to 19 non-zeros per row. Due to the higher condition number the iterative solver takes much longer and even fails to converge on large systems. In contrast, the multigrid solver converges robustly without numerical problems; notice that constructing the multigrid hierarchy is almost the same as for the Laplacian system (up to one more ring of boundary constraints). The computational costs required for the sparse factorization are proportional to the increased number of non-zeros per row. The LU factorization additionally has to incorporate pivoting for numerical stability and failed for larger systems. In contrast, the Cholesky factorization worked robustly in all experiments.

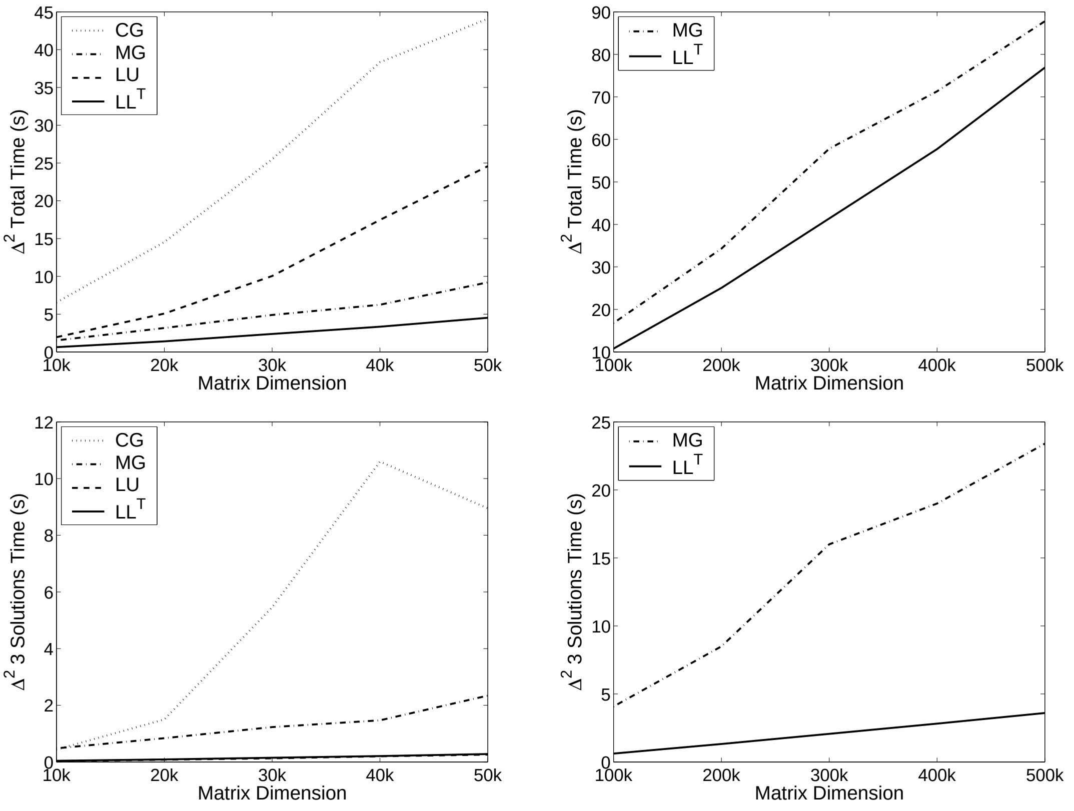
The memory consumption of the multigrid method is mainly determined by the meshes representing the different hierarchy levels. In contrast, the memory required for the Cholesky factorization depends significantly on the sparsity of the matrix, too. On the 500k example the multigrid method and the direct solver need about 1GB and 600MB for the Laplacian system, and about 1.1GB and 1.2GB for the bi-Laplacian system. Hence, the direct solver would not be capable of factorizing Laplacian systems of higher order on current PCs, while the multigrid method would succeed.

These comparisons show that direct solvers are a valuable and efficient alternative to multigrid methods even if the lin-



Size	Iterative CG	Multigrid	*LU*	*LL^T*
10k	0.11/1.56/0.08	0.15/0.65/0.09	0.07/0.22/0.01	0.07/0.14/0.03
20k	0.21/3.36/0.21	0.32/1.38/0.19	0.14/0.62/0.03	0.14/0.31/0.06
30k	0.32/5.26/0.38	0.49/2.20/0.27	0.22/1.19/0.05	0.22/0.53/0.09
40k	0.44/6.86/0.56	0.65/3.07/0.33	0.30/1.80/0.06	0.31/0.75/0.12
50k	0.56/9.18/0.98	0.92/4.00/0.57	0.38/2.79/0.10	0.39/1.00/0.15
100k	1.15/16.0/3.19	1.73/8.10/0.96	0.79/5.66/0.21	0.80/2.26/0.31
200k	2.27/33.2/11.6	3.50/16.4/1.91	1.56/18.5/0.52	1.59/5.38/0.65
300k	3.36/50.7/23.6	5.60/24.6/3.54	2.29/30.0/0.83	2.35/9.10/1.00
400k	4.35/69.1/37.3	7.13/32.5/4.48	2.97/50.8/1.21	3.02/12.9/1.37
500k	5.42/87.3/47.4	8.70/40.2/5.57	3.69/68.4/1.54	3.74/17.4/1.74

Table 1: *Comparison of different solvers for Laplacian systems* $\Delta_{\mathcal{S}}\mathbf{P} = \mathbf{B}$ *of 10k to 50k and 100k to 500k free vertices* **P**. *The three timings for each solver represent matrix setup, pre-computation, and three solutions for the x, y, and z components of* **P**. *The graphs in the upper row show the total computation times (sum of all three columns). The center row depicts the solution times only (3rd column), as those typically determine the per-frame cost in interactive applications.*



Size	Iterative CG	Multigrid	*LU*	*LL^T*
10k	0.33/5.78/0.44	0.40/0.65/0.48	0.24/1.68/0.03	0.24/0.35/0.04
20k	0.64/12.4/1.50	0.96/1.37/0.84	0.49/4.50/0.08	0.49/0.82/0.09
30k	1.04/19.0/5.46	1.40/2.26/1.23	0.77/9.15/0.13	0.78/1.45/0.15
40k	1.43/26.3/10.6	1.69/3.08/1.47	1.07/16.2/0.20	1.08/2.05/0.21
50k	1.84/33.3/8.95	2.82/4.05/2.34	1.42/22.9/0.26	1.42/2.82/0.28
100k	—	4.60/8.13/4.08	2.86/92.8/0.73	2.88/7.29/0.62
200k	—	9.19/16.6/8.50	—	5.54/18.2/1.32
300k	—	17.0/24.8/16.0	—	8.13/31.2/2.07
400k	—	19.7/32.6/19.0	—	10.4/44.5/2.82
500k	—	24.1/40.3/23.4	—	12.9/60.4/3.60

Table 2: *Comparison of different solvers for bi-Laplacian systems* $\Delta_S^2 \mathbf{P} = \mathbf{B}$ *of 10k to 50k and 100k to 500k free vertices* $\mathbf{P}$. *The three timings for each solver represent matrix setup, pre-computation, and three solutions for the components of* $\mathbf{P}$. *The graphs in the upper row again show the total computation times, while the center row depicts the solution times only (3rd column). For the larger systems, the iterative solver and the sparse LU factorization fail to compute a solution.*

ear systems are highly complex. In all experiments the sparse Cholesky solver was faster than the multigrid method, and if the system has to be solved for multiple right-hand sides, the precomputation of a sparse factorization is even more beneficial.

Appendix B: Speaker Biographies

Dr. Olga Sorkine is an Assistant Professor of Computer Science at the Courant Institute of Mathematical Sciences at New York University. She earned her BSc in Mathematics and Computer Science and her PhD in Computer Science from Tel Aviv University (2000, 2006). Following her studies, she received the Alexander von Humboldt Fellowship to pursue postdoctoral research at the Technical University of Berlin, where she stayed for two years. Her research focuses on computer graphics and geometric modeling, with an emphasis on interactive applications. In particular, she is interested in digital content creation tasks, such as shape representation and editing, artistic modeling techniques, computer animation and digital image manipulation. She is the recipient of the EUROGRAPHICS 2008 Young Researcher Award. Olga has extensive experience as a lecturer and teacher, and has presented at various conferences, including SIGGRAPH, EUROGRAPHICS and IMAGINA (as keynote speaker).

Dr. Mario Botsch is a Professor of Computer Science at Bielefeld University. He received his MS in Mathematics from University of Erlangen-Nuremberg in 1999, and then worked for one year at the Max Planck Institute for Computer Science in Saarbruecken. In 2001 he joined the graphics group of RWTH Aachen, from where he received his PhD in 2005. Between 2005 and 2008 he worked as lecturer and senior researcher at ETH Zurich. Since Mai 2008 he is heading the Computer Graphics & Geometry Processing Group at Bielefeld University. His research interests include geometry processing in general, and mesh generation, optimization, and shape editing in particular. He received the EUROGRAPHICS 2007 Young Researcher Award. Mario is an experienced speaker and lecturer, and presented papers and courses at SIGGRAPH, EUROGRAPHICS, and various other international conferences.

References

[ACWK06] ANGELIDIS A., CANI M.-P., WYVILL G., KING S.: Swirling-Sweepers: constant volume modeling. *Graphical Models 68*, 4 (2006).

[AFTCO07] AU O. K.-C., FU H., TAI C.-L., COHEN-OR D.: Handle-aware isolines for scalable shape editing. *ACM Trans. on Graphics (Proc. SIGGRAPH) 26*, 3 (2007).

[AKS05] AKSOYLU B., KHODAKOVSKY A., SCHRÖDER P.: Multilevel Solvers for Unstructured Surface Meshes. *SIAM Journal on Scientific Computing 26*, 4 (2005), 1146–1165.

[AUGA05] ALLIEZ P., UCELLI G., GOTSMAN C., ATTENE M.: Recent advances in remeshing of surfaces. State-of-the-art report, 2005.

[BBC*94] BARRETT R., BERRY M., CHAN T. F., DEMMEL J., DONATO J., DONGARRA J., EIJKHOUT V., POZO R., ROMINE C., DER VORST H. V.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[BBK05] BOTSCH M., BOMMES D., KOBBELT L.: Efficient linear system solvers for geometry processing. In *11th IMA conference on the Mathematics of Surfaces* (2005).

[BD96] BORNEMANN F. A., DEUFLHARD P.: The cascading multigrid method for elliptic problems. *Num. Math. 75*, 2 (1996), 135–152.

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*, 2nd ed. SIAM, 2000.

[BK01] BOTSCH M., KOBBELT L.: A robust procedure to eliminate degenerate faces from triangle meshes. In *Proc. of Vision, Modeling, and Visualization 01* (2001), pp. 283–289.

[BK03a] BENDELS G. H., KLEIN R.: Mesh forging: editing of 3D-meshes using implicitly defined occluders. In *Symposium on Geometry Processing* (2003), pp. 207–217.

[BK03b] BOTSCH M., KOBBELT L.: Multiresolution surface representation based on displacement volumes. In *Proc. of Eurographics 03* (2003), pp. 483–491.

[BK04a] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 630–634.

[BK04b] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proc. of Eurographics symposium on Geometry Processing 04* (2004), pp. 189–196.

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. In *Proc. of Eurographics 05* (2005), pp. 611–621.

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: PriMo: Coupled prisms for intuitive surface modeling. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2006), pp. 11–20.

[BPK*08] BOTSCH M., PAULY M., KOBBELT L., ALLIEZ P., LEVY B., BISCHOFF S., RÖSSL C.: Geometric modeling based on polygonal meshes. In *Eurographics Tutorial Notes* (2008).

[BPWG07] BOTSCH M., PAULY M., WICKE M., GROSS M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum (Proc. Eurographics) 26*, 3 (2007).

[BS07] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics to appear* (2007).

[BSPG06] BOTSCH M., SUMNER R., PAULY M., GROSS M.: Deformation transfer for detail-preserving surface editing. In *Proceedings of Vision, Modeling, and Visualization (VMV)* (2006), pp. 357–364.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proc. of ACM SIGGRAPH 01* (2001), pp. 67–76.

[CG91] CELNIKER G., GOSSARD D.: Deformable curve and surface finite-elements for free-form shape design. In *Proc. of ACM SIGGRAPH 91* (1991), pp. 257–266.

[CM69] CUTHILL E., MCKEE J.: Reducing the bandwidth of sparse symmetric matrices. In *Proc. of the 24th ACM National Conference* (1969), pp. 157–172.

[Coq90] COQUILLART S.: Extended free-form deformation: a sculpturing tool for 3D geometric modeling. In *Proc. of ACM SIGGRAPH 90* (1990), pp. 187–196.

[dC76] DO CARMO M. P.: *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[DEG*99] DEMMEL J. W., EISENSTAT S. C., GILBERT J. R., LI X. S., LIU J. W. H.: A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*, 3 (1999), 720–755.

[DH05] DAVIS T. A., HAGER W.: CHOLMOD: supernodal sparse cholesky factorization and update/downdate. http://www.cise.ufl.edu/research/sparse/cholmod, 2005.

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. of ACM SIGGRAPH 99* (1999), pp. 317–324.

[Duc77] DUCHON J.: Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions of Several Variables*, Schempp W., Zeller K., (Eds.), no. 571 in Lecture Notes in Mathematics. Springer Verlag, 1977, pp. 85–100.

[EDD*95] ECK M., DEROSE T., DUCHAMP T., HOPPE H., LOUNSBERY M., STUETZLE W.: Multiresolution analysis of arbitrary meshes. In *Proc. of ACM SIGGRAPH 95* (1995), pp. 173–182.

[Far97] FARIN G.: *Curves and Surfaces for Computer Aided Geometric Design*, 4th ed. Academic Press, 1997.

[FB88] FORSEY D. R., BARTELS R. H.: Hierarchical B-spline refinement. In *Proc. of ACM SIGGRAPH 88* (1988), pp. 205–212.

[FB95] FORSEY D., BARTELS R. H.: Surface fitting with hierarchical splines. *ACM Transactions on Graphics 14*, 2 (1995), 134–161.

[Flo03] FLOATER M. S.: Mean value coordinates. *Computer Aided Geometric Design 20*, 1 (2003), 19–27.

[GDP*05] GRINSPUN E., DESBRUN M., POLTHIER K., SCHRÖDER P., STERN A.: Discrete differential geometry: An applied introduction. In *SIGGRAPH 2005 Course Notes* (2005).

[GL81] GEORGE A., LIU J. W. H.: *Computer solution of large sparse positive definite matrices*. Prentice Hall, 1981.

[GL89a] GEORGE A., LIU J. W. H.: The evolution of the minimum degree ordering algorithm. *SIAM Review 31*, 1 (1989), 1–19.

[GL89b] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. In *Proc. of ACM SIGGRAPH 99* (1999), pp. 325–334.

[GVSS00] GUSKOV I., VIDIMCE K., SWELDENS W., SCHRÖDER P.: Normal meshes. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 95–102.

[Hac86] HACKBUSCH W.: *Multi-Grid Methods and Applications*. Springer Verlag, 1986.

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 177–184.

[Hor87] HORN B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4 (1987), 629–642.

[HSL*06] HUANG J., SHI X., LIU X., ZHOU K., WEI L.-Y., TENG S., BAO H., GUO B., SHUM H.-Y.: Subspace gradient domain mesh deformation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 25*, 3 (2006), 1126–1134.

[JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (2007), 71.

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (2005), 561–566.

[KCS98] KOBBELT L., CAMPAGNA S., SEIDEL H.-P.: A general framework for mesh decimation. In *Proc. of Graphics Interface 98* (1998), pp. 43–50.

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 105–114.

[KK98] KARYPIS G., KUMAR V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Sci. Comput. 20*, 1 (1998), 359–392.

[KS06] KRAEVOY V., SHEFFER A.: Mean-value geometry encoding. *International Journal of Shape Modeling 12*, 1 (2006), 29–46.

[KVLS99] KOBBELT L., VORSATZ J., LABSIK U., SEIDEL H.-P.: A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum (EG 99 proc.) 18* (1999), 119–130.

[KVS99] KOBBELT L., VORSATZ J., SEIDEL H.-P.: Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory Appl. 14*, 1-3 (1999), 5–24.

[LCOGL07] LIPMAN Y., COHEN-OR D., GAL R., LEVIN D.: Volume and shape preservation via moving frame manipulation. *ACM Transactions on Graphics 26*, 1 (2007).

[Liu85] LIU J. W. H.: Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw. 11*, 2 (1985), 141–153.

[LKCOL07] LIPMAN Y., KOPF J., COHEN-OR D., LEVIN D.: Gpu-assisted positive mean value coordinates for mesh deformations. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 117–123.

[LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. *ACM Trans. Graph. 27*, 3 (2008).

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 85–94.

[LS76] LIU J. W. H., SHERMAN A. H.: Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numerical Analysis 2*, 13 (1976), 198–213.

[LSC*04] LIPMAN Y., SORKINE O., COHEN-OR D., LEVIN D., RÖSSL C., SEIDEL H.-P.: Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International 04* (2004), pp. 181–190.

[LSLC05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. In *Proc. of ACM SIGGRAPH 05* (2005), pp. 479–487.

[LSS*98] LEE A. W. F., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: MAPS: Multiresolution adaptive parameterization of surfaces. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 95–104.

[MBK07] MARINOV M., BOTSCH M., KOBBELT L.: GPU-based multiresolution deformation using approximate normal field reconstruction. *Journal of Graphics Tools* (2007).

[MDSB03] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Hege H.-C., Polthier K., (Eds.). Springer-Verlag, Heidelberg, 2003, pp. 35–57.

[MS92] MORETON H. P., SÉQUIN C. H.: Functional optimization for fair surface design. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 167–176.

[MYC*01] MORSE B. S., YOO T. S., CHEN D. T., RHEINGANS P., SUBRAMANIAN K. R.: Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proc. of Shape Modeling & Applications 01* (2001), pp. 89–98.

[NSAC05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *Proc. of ACM SIGGRAPH 05* (2005), pp. 1142–1147.

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: 3D scattered data approximation with adaptive compactly supported radial basis functions. In *Proc. of Shape Modeling International 04* (2004).

[PFTV92] PRESS W. H., FLANNERY B. P., TEUKOLSKY S. A., VETTERLING W. T.: *Numerical Recipes: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[PKG06] PAULY M., KOBBELT L., GROSS M.: Point-based multi-scale surface representation. *ACM Transactions on Graphics 25*, 2 (2006), 177–193.

[PKKG03] PAULY M., KEISER R., KOBBELT L., GROSS M.: Shape modeling with point-sampled geometry. In *Proc. of ACM SIGGRAPH 03* (2003), pp. 641–650.

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2*, 1 (1993), 15–36.

[RP05] RENARD Y., POMMIER J.: Gmm++: a generic template matrix C++ library. http://www-gmm.insa-toulouse.fr/getfem/gmm_intro, 2005.

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proc. of Eurographics symposium on Geometry Processing* (2007).

[SCL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proc. of Eurographics symposium on Geometry Processing 04* (2004), pp. 179–188.

[SD92] SHOEMAKE K., DUFF T.: Matrix animation and polar decomposition. In *Proc. of Graphics Interface 92* (1992), pp. 258–264.

[SF98] SINGH K., FIUME E.: Wires: A geometric deformation technique. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 405–414.

[She94] SHEWCHUK J. R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., Carnegie Mellon University, 1994.

[SK04] SHEFFER A., KRAEVOY V.: Pyramid coordinates for morphing and deformation. In *Proc. of Symp. on 3D Data Processing, Visualization and Transmission (3DPVT) '04* (2004), pp. 68–75.

[SMW06] SCHAEFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. *ACM TOG 25*, 3 (2006), 533–540.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proc. of ACM SIGGRAPH 86* (1986), pp. 151–159.

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 399–405.

[SSP07] SUMNER R., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Trans. on Graphics (Proc. SIGGRAPH) 26*, 3 (2007).

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 25*, 3 (2006), 1108–1117.

[SZT*07] SHI X., ZHOU K., TONG Y., DESBRUN M., BAO H., GUO B.: Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007).

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proc. of ACM SIGGRAPH 95* (1995), pp. 351–358.

[TCR03] TOLEDO S., CHEN D., ROTKIN V.: Taucs: A library of sparse linear solvers. http://www.tau.ac.il/∼stoledo/taucs, 2003.

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. of ACM SIGGRAPH 87* (1987), pp. 205–214.

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Trans. on Graphics (Proc. SIGGRAPH) 25*, 3 (2006), 1118–1125.

[Wen05] WENDLAND H.: *Scattered Data Approximation*. Cambridge University Press, 2005.

[WMKG07] WARDETZKY M., MATHUR S., KÄLBERER F., GRINSPUN E.: Discrete laplace operators: no free lunch. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007).

[WW92] WELCH W., WITKIN A.: Variational surface modeling. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 157–166.

[XZY*07] XU W., ZHOU K., YU Y., TAN Q., PENG Q., GUO B.: Gradient domain editing of deforming mesh sequences. *ACM Trans. on Graphics (Proc. SIGGRAPH) 26*, 3 (2007).

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 644–651.

[ZG07] ZHU Y., GORTLER S.: *3D Deformation Using Moving least Squares*. Tech. Rep. TR-10-07, Harvard Computer Science, 2007.

[ZHS*05] ZHOU K., HUANG J., SNYDER J., LIU X., BAO H., GUO B., SHUM H.-Y.: Large mesh deformation using the volumetric graph Laplacian. In *Proc. of ACM SIGGRAPH 05* (2005), pp. 496–503.

[ZRKS05] ZAYER R., RÖSSL C., KARNI Z., SEIDEL H.-P.: Harmonic guidance for surface deformation. In *Proc. of Eurographics 05* (2005), pp. 601–609.

[ZSD*00] ZORIN D., SCHRÖDER P., DEROSE T., KOBBELT L., LEVIN A., SWELDENS W.: Subdivision for modeling and animation. In *Course notes of ACM SIGGRAPH 00* (2000).

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proc. of ACM SIGGRAPH 97* (1997), pp. 259–268.