EUROGRAPHICS 2005 / J. Marks and M. Alexa
(Guest Editors)

Volume 24 (2005), Number 3

Real-Time Shape Editing using Radial Basis Functions

Mario Botsch Leif Kobbelt

Computer Graphics Group
RWTH Aachen University

Abstract

Current surface-based methods for interactive freeform editing of high resolution 3D models are very powerful, but at the same time require a certain minimum tessellation or sampling quality in order to guarantee sufficient robustness. In contrast to this, space deformation techniques do not depend on the underlying surface representation and hence are affected neither by its complexity nor by its quality aspects. However, while analogously to surface-based methods high quality deformations can be derived from variational optimization, the major drawback lies in the computation and evaluation, which is considerably more expensive for volumetric space deformations.
In this paper we present techniques which allow us to use triharmonic radial basis functions for real-time freeform shape editing. An incremental least-squares method enables us to approximately solve the involved linear systems in a robust and efficient manner and by precomputing a special set of deformation basis functions we are able to significantly reduce the per-frame costs. Moreover, evaluating these linear basis functions on the GPU finally allows us to deform highly complex polygon meshes or point-based models at a rate of 30M vertices or 13M splats per second, respectively.

1. Introduction

A very popular and important aspect of geometry processing is the interactive deformation of geometric models. In this paper we do not consider the ab-initio creation of models from scratch, but rather the modification of existing models, like those acquired by range scanning or by the tessellation of a CAD model originally represented by NURBS surfaces. Usually the desired target shape is not (exactly) known beforehand, and hence has to be found by exploring different shape deformation options in an interactive manner. As a consequence, a practically useful shape editing method has to be sufficiently fast to allow for real-time deformations even of complex models.

Besides performance, the two other main requirements for shape editing techniques are *exact control* and *high quality* of the deformation. To satisfy the first requirement the deformation method has to be able to incorporate arbitrary displacement constraints $\mathbf{p}_i \mapsto \mathbf{p}_i'$, which map a point $\mathbf{p}_i$ to its desired target position $\mathbf{p}_i'$. Obviously this also allows to exactly prescribe the support of the modification by mapping all fixed vertices $\mathbf{f}_i$ outside the support region onto themselves: $\mathbf{f}_i \mapsto \mathbf{f}_i$.

High quality deformations should meet these constraints and otherwise be free of unnecessary oscillations, following the principle of *simplest shape* for fair surface generation [Sap94]. In that context, smooth or *fair* thin-plate-like surfaces are derived by a constrained variational optimization of some curvature energy functional [MS92, WW92]. Fair deformations are analogously computed as the difference of two fair surfaces $\mathcal{S}$ and $\mathcal{S}'$, i.e., as a fair displacement function $\mathbf{d} : \mathcal{S} \rightarrow \mathcal{S}'$. This deformation function is controlled by adjusting the boundary constraints of the optimization, which is why approaches of this kind are called *boundary constraint modeling* (BCM).

Most existing BCM approaches are *surface-based*, i.e., they can be thought of as computing a fair deformation field *on* the surface $\mathcal{S}$. If the underlying surface representation is a triangle mesh, computing the deformation field usually requires to solve a linear Laplacian system on $\mathcal{S}$. 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.

This tight connection unfortunately also prevents the derivation of a uniform deformation framework for several

types of surface representations. Here, point-sampled geometries [ZPvBG01] are a particularly interesting alternative to triangle meshes, since they provide the same approximation power, but offer additional flexibility, since individual splats do not have to be connected in a C^0 manner [KB04].

The above problems are avoided by volumetric *space deformation* techniques, that compute a tri-variate deformation function $\mathbf{d} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, which is used to transform all points of the original surface $\mathcal{S}$ to the modified surface $\mathcal{S}' = \{\mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\}$. Since the deformation does not depend on a particular surface representation, this uniform deformation framework can be applied to all explicit surface representations, e.g., by transforming all vertices of a triangle mesh or all splat centers of a point-based model.

In this paper we use *radial basis functions* (RBF) for volumetric boundary constraint modeling, as they provide maximum flexibility w.r.t. constraint or control point placement, as well as high quality deformations of provable fairness when using the triharmonic radial basis function $\varphi(r) = r^3$. The problem of this optimal basis function is that the linear system required for computing the deformation is dense and therefore difficult to solve. Because of this, most existing approaches use simpler compactly supported basis functions and trade superior fairness for computational efficiency.

Due to their high quality we nevertheless advocate for the use of triharmonic radial basis functions and derive the necessary techniques to overcome their computational restrictions and allow for real-time deformations. In Sect. 3 we first present a modeling metaphor for setting up the deformation constraints for the linear RBF system (Sect. 4). In order to be able to efficiently solve the resulting dense linear systems, we introduce an incremental least-squares solver in Sect. 5.

The high per-frame costs for computing and applying a deformation can be significantly reduced by precomputing a set of linear basis functions for the deformation (Sect. 6). Additionally using the deformation's Jacobian enables the individual transformation of each point and its normal vector. When implemented as a vertex shader on modern GPUs, this technique provides real-time deformation of up to 30M vertices or 13M surface splats per second (Sect. 7).

We will show in Sect. 8 that our space deformation technique can efficiently be applied even when either the mesh quality, the mesh complexity, non-manifold configurations, or the surface representation in general prevent the use of surface-based deformation methods. However, we have to emphasize that if the input model allows for both space- and surface-based approaches, the latter usually provide more fine-grained control of boundary constraints, like the segment-wise specification of boundary continuities. Moreover they enable geodesically anisotropic bending and more plausible detail preservation under extreme deformation [BK04a]. However, our method outperforms existing space- and surface-based modeling approaches in terms of frame rates due to its efficient GPU implementation.

2. Related Work

In order to efficiently compute a deformation field of minimal curvature-energy, surface-based BCM approaches use variational calculus to derive a PDE that is then solved for the optimal deformation function. This PDE is discretized to a large sparse linear (bi-)Laplacian system, which is solved for (the displacements of) all free vertices [KCVS98, BK04a, LSCO*04, SCOL*04, YZX*04].

Since during a modeling session this linear system has to be solved each time the user interactively changes the constraints, efficient sparse solvers of linear time complexity are required [BBK05], or special deformation basis functions have to be precomputed [BK04a]. Notice that in the presence of degenerate triangles the discrete Laplacian operator is not defined and the linear system becomes singular. In this case quite some effort has to be spent to still be able to compute fair deformations for the numerically ill-conditioned meshes, like eliminating degenerate triangles [BK01] or even remeshing the complete surface [BK04b].

Extending these mesh-based approaches to point-sampled geometries is not straightforward, since the missing neighborhood relation considerably complicates the generalization of the Laplacian operator to this surface representation [CRT04]. In their shape modeling approach, Pauly et al. [PKKG03] therefore chose a simpler distance-based deformation propagation, thereby trading provably high deformation quality for simpler and more efficient computations.

Space deformation techniques avoid these problems, because they implicitly modify objects by deforming their embedding space. As a consequence, these methods are influenced neither by the complexity nor by the quality of a surface tessellation.

The classical freeform deformation (FFD) method [SP86] and its variants [Coq90, MJ96] represent the space deformation by a tensor-product spline function, which requires complex user-interactions and might cause aliasing problems, as described in [BK04a]. In order to satisfy given displacement constraints, the inverse FFD method [HHK92] solves a linear system for the lattice deformation. This system may be over- as well as under-determined and hence is solved by least-squares or least-norm methods, respectively. The latter, however, minimizes the amount of control point movements, which does not necessarily imply a fair deformation of low curvature energy.

Other approaches deform a so-called *control handle* (a point, curve, or surface region) and propagate its transformation into its Euclidean or geodesic vicinity [SF98, BK03a, PKKG03]. The two-handed modeling interface Twister [LKG*03] also falls into this category. In contrast to boundary constraint modeling approaches, these methods avoid the per-frame solution of a linear system and are therefore highly efficient, but in consequence also lose global optimality properties like curvature energy minimiza-

tion. In addition, these methods may fail to smoothly interpolate (the displacement of) several independently controlled and arbitrarily shaped handles.

Radial basis functions (RBFs) are commonly used for all kinds of scattered data interpolation problems, since they are able to interpolate arbitrary constraints in a smooth manner, like for instance fitting a scalar-valued signed distance function to the given sample points and their normals [SPOK95, CBC*01, MYC*01, OBS04, TRS04]. A trivariate scalar RBF is defined by a set of centers $\mathbf{c}_j \in \mathbb{R}^3$ and weights $w_j \in \mathbb{R}$ as

$$\begin{aligned} f(\mathbf{x}) &= \sum_j w_j \varphi\left(\left\|\mathbf{c}_j - \mathbf{x}\right\|\right) + p(\mathbf{x}) \\ &= \sum_j w_j \varphi_j(\mathbf{x}) + p(\mathbf{x}) \quad , \end{aligned} \tag{1}$$

where $\varphi_j(\cdot) = \varphi\left(\left\|\mathbf{c}_j - \cdot\right\|\right)$ is the basis function corresponding to the jth center $\mathbf{c}_j$ and $p(\mathbf{x})$ is a polynomial of low degree used to guarantee polynomial precision.

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, OBS03, 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 $p(\cdot) \in \Pi_2$, the function (1) is triharmonic ($\Delta^3 f = 0$) and hence minimizes the energy

$$\|f\|^2 = \int_{\mathbb{R}^3} f_{xxx}^2(\mathbf{x}) + f_{xxy}^2(\mathbf{x}) + \ldots + f_{zzz}^2(\mathbf{x})\, d\mathbf{x} \quad .$$

These trivariate functions are conceptually equivalent to the minimum variation surfaces of [MS92] and the triharmonic surfaces used in [BK04a].

Due to the global support of this RBF the resulting linear system (see Sect. 4) is dense and the cubic complexity for solving it limits these methods to a few thousand sample points. Carr et al. [CBC*01] use a fast multi-pole evaluation method to derive an efficient iterative solver with linear time complexity, such that globally supported basis functions can be used even for highly complex point sets. Unfortunately, the implementation of their method is very complicated and only commercially available.

First shape modeling approaches based on RBFs define the original surface $\mathcal{S}$ as an RBF interpolant of a given set of point and normal constraints, and modify the surface by changing these interpolation constraints [TO02, RTSD03]. Both methods use globally supported basis functions and are therefore limited to a small number of constraints, restricting them to smooth blobby surfaces without fine surface details or sharp features.

Since our goal is not a smooth surface, but rather the smooth deformation of a given surface, the more promising way is to represent the space deformation function (instead of the surface itself) by a vector-valued RBF

$$\mathbf{d}(\mathbf{x}) = \sum_j \mathbf{w}_j \varphi_j(\mathbf{x}) + \mathbf{p}(\mathbf{x}) \quad , \tag{2}$$

where the weights $\mathbf{w}_j \in \mathbb{R}^3$ are computed to smoothly interpolate a given set of displacement constraints. To our knowledge, all existing methods of this kind use compactly supported basis functions in order to achieve faster response times [BR94, KSSH02], which, in turn, limits their fairness and additionally restricts the range of possible deformations, because a fixed support radius for the basis functions has to be prescribed upfront.

Because of their superior fairness we propose to use globally supported triharmonic radial basis functions. In the following we therefore present techniques that allow us to use these functions for real-time shape editing even for complex surfaces and complex deformations.

3. Modeling Metaphor

The design of the modeling metaphor is crucial for a practically useful shape editing framework, since it is responsible for translating the deformation the designer has in mind into the boundary constraints of the variational optimization, i.e., into a set of displacement constraints $\mathbf{p}_i \mapsto \mathbf{p}_i'$, which is then to be smoothly interpolated by a radial basis function. Our user interface closely follows the intuitive BCM approach presented in [KCVS98, BK04a], but additionally extends it by non-rigid handle curves.

The *support* of the modification can be an arbitrary surface region, i.e., a set of vertices $\{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$, and is specified by the user by drawing it onto the surface. Within this region a *control handle* is selected as another set of points $\{\mathbf{h}_1, \ldots, \mathbf{h}_h\}$, which is then transformed by specifying an affine mapping $\mathbf{m}(\cdot)$ using some kind of manipulator widget, yielding $\mathbf{h}_i' := \mathbf{m}(\mathbf{h}_i)$. Having the surface partitioned into support vertices $\mathbf{p}_i$, handle vertices $\mathbf{h}_i$, and the remaining fixed vertices $\mathbf{f}_i$, the displacement constraints are $\mathbf{h}_i \mapsto \mathbf{h}_i'$ and $\mathbf{f}_i \mapsto \mathbf{f}_i$. These displacements are smoothly interpolated by a RBF $\mathbf{d}(\cdot)$ as described in the next section, and finally all points within the support region are transformed by it: $\mathbf{p}_i' = \mathbf{d}(\mathbf{p}_i) \in \mathcal{S}'$.

The triharmonic basis function $\varphi(r) = r^3$ guarantees a fair deformation function $\mathbf{d}(\cdot)$ without unnecessary oscillations, which is able to interpolate C^2 boundary constraints. In order to achieve a smooth connection of the support region with the transformed handle and the fixed part of the surface, *surface-based* approaches specify (approximate) C^2 constraints for triharmonic surfaces by three rings of constrained vertices [BK04a] (cf. Fig. 1, left).

Analogously, it is sufficient to specify the constraints of a *space deformation* by three rings of vertices, or, geometrically equivalent, by a band of three points thickness along

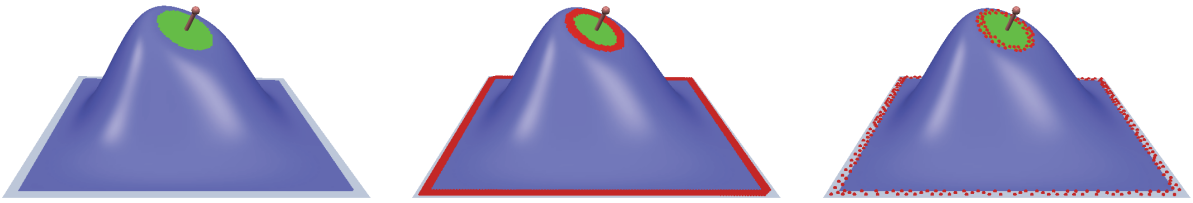

Figure 1: *The blue support region is deformed by smoothly interpolating the affinely transformed green control handle. The fair triharmonic surface-based deformations of [BK04a] (*left*) can be reproduced by the triharmonic space deformation, where the C^2 constraints are defined by the red bands of three points thickness (*center*). However, the number of centers required for a sufficiently accurate approximation (see Sect. 5) is usually significantly lower, like 20% in this example (*right*).*

the support's boundary: Interpolating one boundary contour in a C^2 Hermite manner is conceptually equivalent to interpolating three nearby offset contours, since the latter corresponds to a finite difference approximation of the first and second derivatives, which is sufficiently accurate because the RBF displacement function is smooth (cf. Fig. 1, center).

This simple and intuitive *rigid control handle* interface turned out to be sufficiently powerful for most modifications, especially since more complicated deformations can be decomposed into a sequence of simpler ones. However, in some CAD systems the bending behavior of a surface is intuitively controlled by specifying and modifying curves *on* the surface. Such *non-rigid control curves* can easily be integrated into our framework: An initial spline curve $\mathbf{c}(t)$ is constructed by interpolating a set of selected points on the surface $\mathcal{S}$, and is then deformed to $\mathbf{c}'(t)$ by moving its spline control points. A sufficiently dense sampling of both $\mathbf{c}$ and $\mathbf{c}'$ then yields the required displacement constraints:

$$\mathbf{h}_i := \mathbf{c}(t_i) \;\mapsto\; \mathbf{c}'(t_i) =: \mathbf{h}_i' \quad .$$

This control curve metaphor is particularly suited for space deformations, as in this case the curve does not have to lie exactly on the surface and the constraint points $\mathbf{c}(t_i)$ are not restricted to vertex positions, as they usually are for surface-based approaches (cf. Fig. 3, left).

4. RBF Interpolation

After setting up the displacement constraints for the handle points and fixed points as described in the last section, an RBF deformation function (2) interpolating the constraints $\mathbf{d}(\mathbf{f}_i) = \mathbf{f}_i$ and $\mathbf{d}(\mathbf{h}_i) = \mathbf{h}_i'$ has to be found. Combining these constraints into one set $\mathbf{d}(\mathbf{x}_i) = \mathbf{b}_i$, for $1 \le i \le m$, and selecting the RBF centers as $\mathbf{c}_i := \mathbf{x}_i$ leads to the symmetric linear system

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{w}_j \\ \mathbf{q}_j \end{pmatrix} = \begin{pmatrix} \mathbf{b}_i \\ 0 \end{pmatrix} \quad , \tag{3}$$

where $\Phi \in \mathbb{R}^{m\times m}$ and $P \in \mathbb{R}^{m\times 10}$ are defined by $\Phi_{ij} = \varphi_j(\mathbf{c}_i)$ and $P_{ij} = p_j(\mathbf{c}_i)$, and $\{p_1,\ldots,p_{10}\}$ is a basis of the space of trivariate quadratic polynomials Π_2. This system is solved for the vector-valued RBF weights $\mathbf{w}_1,\ldots,\mathbf{w}_m$ and the vector-valued coefficients $\mathbf{q}_1,\ldots,\mathbf{q}_{10}$ of the quadratic polynomial, resulting in the deformation function $\mathbf{d}(\cdot)$.

The matrix P and hence the complete system is singular if all constraints $\mathbf{c}_i$ lie on a quadric [Mic86]. In such situations we simply omit the polynomial, which turned out to not have a large influence and still leads to high quality deformations (cf. Fig. 1). For the sake of a simpler notation we omit the polynomial in all following discussions and focus on the upper left $m \times m$ block only, which we denote as

$$\Phi \cdot W = \begin{pmatrix} F \\ H' \end{pmatrix} \tag{4}$$

with weights $W = (\mathbf{w}_1,\ldots,\mathbf{w}_m)^T$, fixed vertices $F = (\mathbf{f}_1,\ldots,\mathbf{f}_f)^T$, and handle vertices $H' = (\mathbf{h}_1',\ldots,\mathbf{h}_h')^T$. The generalization to the full system (3) is straightforward.

Surface-based approaches solve significantly larger linear systems for all free vertices $\mathbf{p}_1,\ldots,\mathbf{p}_N$, but due to their sparsity, these systems can be solved with complexity $O(N)$ [BBK05]. In contrast, the above system is solved for a relatively small number of weights $\mathbf{w}_1,\ldots,\mathbf{w}_m$ only, with m usually being of the order $\sqrt{N}$ (assuming uniform sampling density). Since $\varphi(r) = r^3$ is globally supported, the $m \times m$ system (4) is dense and its solution has cubic complexity $O(m^3) = O(N^{1.5})$, resulting in a slightly worse overall computational complexity of the RBF approach. We therefore improve the performance by using the incremental solver presented in the next section.

5. Incremental Least Squares Solver

Since the space deformation $\mathbf{d}(\cdot)$ is — up to the constraint specification — independent of the surface tessellation, we can expect the computational costs to be mainly determined by the geometric complexity of the deformation itself, rather than by the resolution of the tessellation: A simple deformation of a highly over-tessellated mesh should still be simple to compute. This is reflected by the observation that we usually do not need all basis functions $\varphi_1,\ldots,\varphi_m$ in order to solve (4) up to a sufficiently small approximation error.

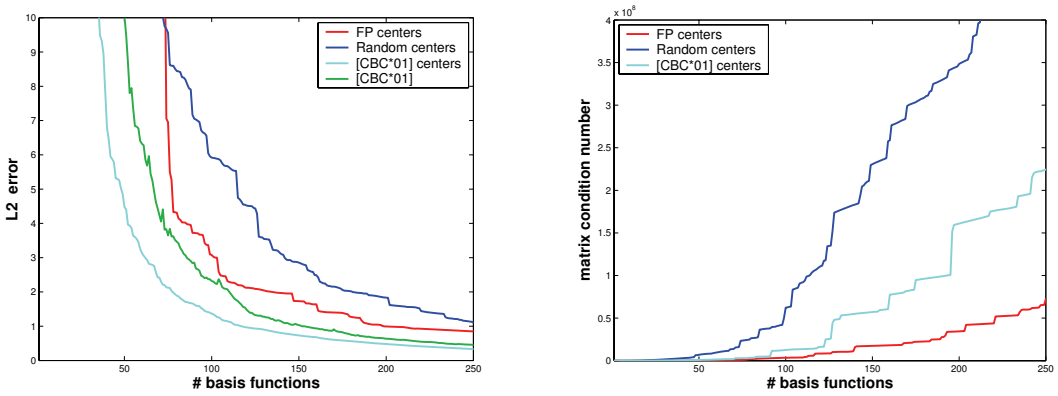

Figure 2: *Comparing L_2 error (*left*) and matrix condition (*right*) of incremental approximation methods, which differ in their center selection strategy and approximation technique. The method of [CBC*01] leads to a non-monotonically decreasing error (*green*), that is worse than an L_2-optimal least-squares fit using the* same *RBF centers (*cyan*). Since the expensive computation of these centers optimizes for one particular right-hand side only, we use random selection (*blue*) or farthest point sampling (*red*) in order to select the next center. Since the latter strategy optimizes for linearly independent columns, the condition number of the resulting matrix is clearly superior to the other approaches (*right*).*

In [CBC*01] the same observation led to an incremental method for fitting an implicit function to a dense set of surface samples. Starting from a few basis functions, they incrementally improve the approximation by adding more and more centers (respectively basis functions) at the samples with maximum error. In order to enforce *exact* interpolation at the selected centers $\mathbf{c}_1,\ldots,\mathbf{c}_n$, they solve a $n \times n$ system corresponding to the upper left block of (4) in each refinement iteration. Since the samples are assumed to correspond to a smooth surface, the approximation error at the remaining constraints $\mathbf{c}_{n+1},\ldots,\mathbf{c}_m$ is expected to also decrease.

However, it is known that for a prescribed number of basis functions $\varphi_1,\ldots,\varphi_n$, a better global error distribution can be achieved by an optimal L_2 approximation considering *all* constraints $\mathbf{c}_1,\ldots,\mathbf{c}_m$ (cf. Fig. 2, left). This requires solving the over-determined system corresponding to the left $m \times n$ block of (4) in the least-squares sense, which is most robustly performed using the QR factorization [GL89].

If we want to incrementally build and refine an RBF approximation by selecting more and more basis functions until a prescribed L_2 error is satisfied, two questions arise: which basis functions to select, and how to efficiently check for the current approximation error. The overall computation time of the incremental method obviously should be lower than the solution of the full $m \times m$ system. As a consequence, instead of solving at iteration n a new $m \times n$ least-squares system from scratch, the computations from previous iterations should be re-used.

Taking a closer look at the QR factorization [GL89] will reveal in the following that this method can be adjusted to incrementally solve a given least-squares system column by column without introducing any noticeable overhead. Adding one more basis function φ_i is equivalent to appending one more column to the least-squares system, which we will show corresponds to one further iteration of an incremental QR solver.

For an over-determined $m \times n$ system $Ax = b$, the right-hand side will in general not lie in the range of A, i.e., $b \notin \mathrm{rg}(A)$, and hence the system cannot be solved exactly. The optimal point $x^* \in \mathbb{R}^n$, which minimizes the L_2 norm of the residual $r = Ax^* - b$, is characterized by $y = Ax^*$ being the orthogonal projection of b onto $\mathrm{rg}(A)$. In the restricted and full QR factorization

$$A \;=\; Q_1 R \;=\; (Q_1\; Q_2)\begin{pmatrix} R \\ 0 \end{pmatrix}$$

the columns of $Q_1 \in \mathbb{R}^{m\times n}$ provide an orthogonal basis of $\mathrm{rg}(A)$ required for this projection, such that $y = Q_1 Q_1^T b$ and $x^* = R^{-1} Q_1^T b$.

Since the combined matrix $Q := (Q_1\; Q_2) \in \mathbb{R}^{m\times m}$ is orthogonal, b can be represented as $b = Q_1 Q_1^T b + Q_2 Q_2^T b$ and the residual error is

$$\begin{aligned} \left\|b - Ax^*\right\| \;&=\; \left\|b - Q_1 Q_1^T b\right\| = \left\|Q_2 Q_2^T b\right\| \\ &=\; \left\|Q_2^T b\right\| = \left\|(Q^T b)_{(n+1):m}\right\|, \end{aligned}$$

where we exploit the orthogonality of Q_2 and denote by $(x)_{(n+1):m}$ the vector $(x_{n+1},\ldots,x_m)^T$.

The numerically most robust way to compute the QR factorization builds up Q iteratively as a product of orthogonal Householder reflections $Q^T = H_m^T \cdots H_2^T H_1^T$ by processing A column by column. During this process $Q^T b$ is automatically computed by multiplying b with the same sequence

of Householder reflections, such that the error in the current iteration can be computed as $\|(Q^T b)_{(n+1):m}\|$ without any additional overhead and without even computing the approximate solution x^*.

The implementation of an incremental version of the QR factorization is straightforward and requires only a slight reordering of operations when starting from a standard QR factorization algorithm. The steps performed at each iteration n are sketched as follows:

For $n = 1$ to m do:

1. Select or construct A's next column a_n.
2. Multiply a_n with all previous Householder reflections: $a_n \leftarrow H_{n-1} \cdots H_1 a_n$.
3. Compute next Householder reflection $H_n = H(a_n)$ and apply it to $a_n \leftarrow H_n a_n$ and $b \leftarrow H_n b$.
4. Break if $\left\| b_{(n+1):m} \right\| < \varepsilon$.

Solve triangular $n \times n$ system $Rx = b$.

After exiting the loop, the matrix $R = Q^T A$ has been stored in the upper triangle entries of A, and b has been overwritten by $Q^T b$. As the Householder reflections can be stored in A's lower triangle, the only additional storage is a n-vector holding the diagonal elements.

When solving the system for several right-hand sides b_k, the QR factorization can be re-used, but may have to be refined depending on the L_2 error w.r.t. the new right-hand side. Notice that adding a new column a_i monotonically decreases the error for *any* right-hand side b_k, since this corresponds to enlarging the space $\mathrm{rg}(A)$ by one dimension, which allows for a better choice of $y = Ax^*$ (cf. Fig. 2, left).

The missing component is a strategy for selecting the basis function to be added in the next iteration. Since we want to re-use the QR factorization to solve the system for several right-hand sides, we do not try to find the next basis function φ_j that would minimize the residual error for one particular right-hand side. We can, however, choose the basis functions in a way that optimizes the numerical condition of the resulting matrices A and R.

In the presence of (almost) linearly dependent columns the matrix (4) will become singular, therefore we should prefer columns that are linearly independent. Since one column corresponds to the sampling of a basis function φ_j at all centers $\mathbf{c}_1, \ldots, \mathbf{c}_m$, two columns a_j and a_k are (numerically) dependent if their corresponding centers $\mathbf{c}_j$ and $\mathbf{c}_k$ are too close to each other. As a consequence, a farthest point sampling, which in each iteration selects the basis function with the center having the maximum distance to the already selected ones, will yield a uniform sampling and maximally linearly independent columns, resulting in a numerically well-conditioned matrix (cf. Fig. 1, right, and Fig. 2). Since the number m of centers $\mathbf{c}_i$ is rather small, this farthest point re-ordering can be computed efficiently (below 0.2s in all examples).

As the incremental version of the QR factorization does not introduce any overhead besides the error checking, a full incremental factorization using all columns of (4) takes about the same time as the standard QR factorization. However, in all our experiments significantly fewer basis functions $n \ll m$ had to be used to yield results equivalent to the exact solution. The approximation quality is controlled by prescribing a sufficiently small average relative L_2 error

$$\frac{\|Ax - b\|}{m\,\|b\|} \;=\; \frac{\left\| (Q^T b)_{(n+1):m} \right\|}{m\,\|b\|} \;<\; \varepsilon \;.$$

Since the overall complexity of the least-squares method is quadratic in the number of basis functions n, and the later evaluation at all free vertices $\mathbf{p}_i$ is linear, the incremental method allows for a significant acceleration by reducing the required n.

6. Basis Function Precomputation

The incremental QR factorization allows for an efficient (approximate) solution of the linear system (4). During a modeling session, this system has to be solved each time the user updates the constraints, i.e., moves the control handle or changes the control curve (*fitting*). The resulting deformation function $\mathbf{d}(\cdot)$ is then used to map all original support vertices $\mathbf{p}_i \in \mathcal{S}$ to the new $\mathbf{p}'_i \in \mathcal{S}'$ (*evaluation*).

Although the factorization can be re-used, the required per-frame costs are too high to allow for an interactive editing of complex models. Notice that both the fitting and the evaluation process can be the bottleneck, depending on the number of radial basis functions n and the number of support vertices N. In order to minimize the per-frame costs, we extend the idea of precomputed basis functions of [BK04a] to our RBF deformations. We can write the approximate solution of (4) as

$$W \;=\; \Phi_n^+ \begin{pmatrix} F \\ H' \end{pmatrix},$$

where Φ_n^+ represents the least-squares pseudo-inverse of the left $m \times n$ block of (4) [GL89]. Notice that F stays constant during a deformation, and that the handle vertices H are only affinely transformed and hence can be represented as an affine combination

$$H \;=\; M\,(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})^T \;=:\; MC$$

using a matrix $M \in \mathbb{R}^{m \times 4}$ of affine coordinates w.r.t. a local coordinate frame defined by four control points $C = (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})^T \in \mathbb{R}^{4 \times 3}$. Moving the handle changes C to $C' = \mathbf{m}(C)$, such that $H' = MC'$. Exploiting this, the above system for computing the weights W simplifies to

$$W \;=\; \Phi_n^+ \begin{pmatrix} F \\ 0 \end{pmatrix} \;+\; \Phi_n^+ \begin{pmatrix} 0 \\ M \end{pmatrix} C'.$$

The evaluation of the deformation at all free points $P = (\mathbf{p}_1, \ldots, \mathbf{p}_N)$ is a linear operator in W and can be written as $P' = \Phi_N W$ with $(\Phi_N)_{ij} = \varphi_j(\mathbf{p}_i)$. Since the new points P' depend linearly on W, they are consequently also linear in C' and can be computed as

$$P' = \underbrace{\Phi_N \Phi_n^+ \begin{pmatrix} F \\ 0 \end{pmatrix}}_{=:B_F} + \underbrace{\Phi_N \Phi_n^+ \begin{pmatrix} 0 \\ M \end{pmatrix}}_{=:B} C'.$$

The matrices $B_F \in \mathbb{R}^{N\times 3}$ and $B \in \mathbb{R}^{N\times 4}$ can be precomputed and represent a linear basis function for the deformation. When written in terms of displacement vectors ($\delta C = C' - C$), this formula simplifies to

$$P' = P + B\,\delta C.$$

Hence, the per-frame fitting and evaluation of $\mathbf{d}(\cdot)$ can be replaced by a weighted sum of four frame displacements $\delta\mathbf{a}$, $\delta\mathbf{b}$, $\delta\mathbf{c}$, and $\delta\mathbf{d}$ for each point $\mathbf{p}_i$. The precomputation of B requires four solutions of (4) with the columns of M as right-hand sides and the evaluation of the resulting RBF at the points $\mathbf{p}_i$. If several rigid control handles are used, then for each of them a basis function $B_i \in \mathbb{R}^{N\times 4}$ can be computed analogously. Notice that also for this precomputation the incremental QR solver is crucial, as it reduces the computation time from minutes to a few seconds (cf. Table 1).

For the above derivation we exploited the fact that all handle points H can be represented as an affine combination $H = MC$ of a few control points C and that the corresponding affine coordinates M stay constant during the deformation. Notice that the same does also hold for the control curve metaphor introduced in Sect. 3: For spline curves, all samples

$$\mathbf{c}(t_i) = \sum_{j=0}^{k} \mathbf{b}_j B_j^k(t_i)$$

are actually an affine combination of the curve's control points $\mathbf{b}_j$, with the weights given by the B-spline basis functions $B_j^k(t_i)$. As a consequence, we can precompute the same kind of linear basis functions for control curves, just the number of rows of C and columns of M and B changes to $k+1$ for a curve defined by $k+1$ control points.

7. GPU Implementation

The precomputed basis functions allow for efficient shape editing at a rate of about 1.5M vertices per second. Since the pure vertex transformation $\mathbf{p}_i' = \mathbf{d}(\mathbf{p}_i)$ is now sufficiently fast, other factors become the bottleneck, like updating per-face and per-vertex normal vectors for triangle meshes or recomputing the tangent axes for point-based models.

However, we can exploit the fact that the Jacobian $J_{\mathbf{d}}(\cdot) \in \mathbb{R}^{3\times 3}$ of the deformation function $\mathbf{d}(\cdot)$ can be computed analytically. It is well known that a point and its normal vector can be transformed by a deformation and its inverse transposed Jacobian, i.e.,

$$\mathbf{p}_i' = \mathbf{d}(\mathbf{p}_i) \quad \text{and} \quad \mathbf{n}_i' = J_{\mathbf{d}}(\mathbf{p}_i)^{-T}\,\mathbf{n}_i \ .$$

Equivalently, the tangent axes of a surface splat are deformed by the Jacobian itself, resulting in proper rotations and anisotropic stretching of splats (cf. Fig. 3).

It seems prohibitively expensive to evaluate and invert the Jacobian at each point $\mathbf{p}_i$, but using exactly the same ideas as presented in the last section, we can precompute basis functions for $J_{\mathbf{d}}$ as well. For this we replace the evaluation of $\mathbf{d}(\cdot)$, i.e., the matrix Φ_N, by the evaluation of its partial derivatives $\mathbf{d}_x$, $\mathbf{d}_y$, and $\mathbf{d}_z$, yielding the three $N \times 4$ basis function matrices B_x, B_y, and B_z.

Notice that using basis functions for $\mathbf{d}(\cdot)$ as well as for $J_{\mathbf{d}}(\cdot)$ allows us to *individually* process each point with its associated normal vector or tangent axes, since there is no need to re-compute this derivative information from a (transformed) neighborhood of vertices. As a consequence, all per-vertex deformation computations can now be delegated to the GPU by deriving a simple vertex shader for transforming points and normals.

The required input for the shader program are the basis functions, i.e., the respective rows of B, B_x, B_y, and B_z, which are passed as texture coordinates, and the displacement of the control frame δC, which is the same for all vertices and is passed in a global shader variable. Using this setup, all per-vertex attributes like original position, normal vector, and basis functions do not change during the interactive shape editing process. Hence, this static data can be stored in more efficient GPU or AGP memory, which minimizes data transfer costs and is a common optimization for high performance rendering. Only a new frame δC has to be transferred for each frame.

Since each vertex has to be transformed several times for rendering all of its incident triangles, we employ a simple greedy triangle re-ordering to better exploit the GPU's vertex cache [Hop99], followed by a vertex re-ordering to minimize GPU memory cache misses. This simple optimization reduces the average number a vertex has to be processed to about 1.3–1.5, which can improve the performance by a factor of 2–3, depending on the initial triangle ordering of the model.

Due to the high and steadily increasing streaming performance of today's GPUs, delegating the complete geometry deformation to a vertex shader is more than one order of magnitude faster than evaluating the basis functions on the CPU. Notice that for deformations using several independently controlled handle regions or handle curves, the user can only manipulate one handle at a time, such that only the basis functions corresponding to the currently active handle have to be uploaded to the GPU, thereby saving GPU memory and reducing transfer costs even further.

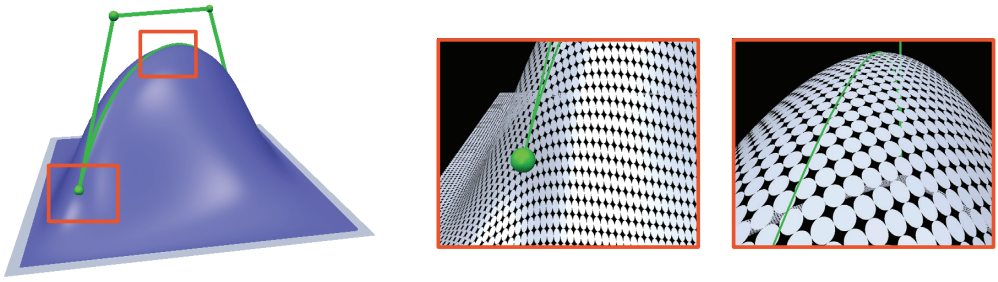

Figure 3: *Non-rigid control curves are an intuitive and powerful metaphor for controlling the surface's bending behavior (*left*), which can be applied to both polygon meshes and point sets. When applying a volumetric deformation to a set of elliptical splats, the Jacobian of the deformation function can be used to compute transformed normal vectors or tangent axes. This allows for exact and hole-free editing of point-sampled geometries. The correct anisotropic stretching can clearly be noticed for the down-scaled splats shown in the closeup images on the right.*

8. Results & Discussion

In this section we quantify the performance gains on complex meshes achieved by the incremental least-squares solver, the precomputed basis functions, and the GPU-based implementation. All timings we give were taken on a 3.0GHz Pentium4 machine, equipped with a nVIDIA GeForce 6800 Ultra GPU, and running Linux.

The deformations depicted in Figs. 4–7 show the flexibility of our deformation framework; more examples can be found in the accompanying video. The timings for computing these deformations using the different techniques proposed in this paper are given in Table 1. It can be seen that a naïve solution of the full system (4) (i.e., using all RBF centers $\mathbf{c}_i$) is up to two orders of magnitude slower than the incremental QR solver, for which a sufficiently small error tolerance of 10^{-7} guaranteed comparable results.

The incremental QR solver also allows for an efficient precomputation of basis functions for $\mathbf{d}(\cdot)$ and its Jacobian $J_{\mathbf{d}}(\cdot)$. These in turn enable interactive shape editing, as they reduce the per-frame computation time by another order of magnitude. Moreover, delegating the complete computation to the GPU improves performance by a further order of magnitude, providing real-time deformations at 30fps of complex models consisting of 2M triangles.

Since space deformations are independent of the surface representation, the same framework can also be used to deform point-sampled geometries. Transforming the splats' tangent axes by $J_{\mathbf{d}}(\cdot)$ correctly stretches splats and retains a hole-free surface representation (cf. Fig. 3). Our GPU-based deformation integrates seamlessly with current hardware accelerated point-based rendering methods [BK03b, KB04]. However, since for high quality visualization two rendering passes are required, the effective frame rate reduces to 10M–13M splats per second.

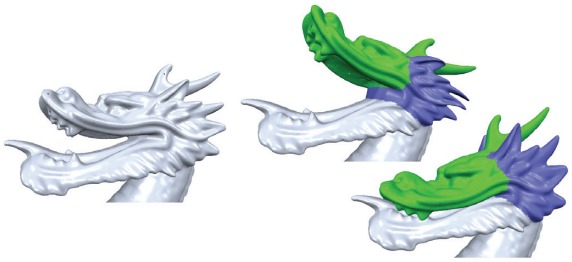

Figure 4: *Opening and closing the mouth of the Dragon. The holes and degenerate triangles contained in this model are no problem for our RBF space-deformation method.*

Figure 5: *A bowing deformation of a scanned Chinese statue by bending its back and neck.*

Model	Vertices	Support	LU	IQR	Basis Precomp.	CPU	GPU
Dragon	437k	36k	31.6 (2352)	10.2 (1310)	17.8 (1680)	0.127	0.018
Bunny	557k	120k	212 (4913)	2.54 (280)	4.9 (340)	0.254	0.018
Male	332k	270k	153 (4136)	1.99 (160)	7.5 (290)	0.271	0.011
Statue	1M	355k	307 (5336)	4.01 (220)	7.6 (230)	0.537	0.032
Bust	984k	880k	215 (4129)	4.68 (130)	16.0 (130)	0.794	0.030

Table 1: *Timings (in seconds) for computing and rendering shape deformations on a range of different models, whose complexities and numbers of active (blue) vertices is given in columns 2 and 3. Computing a deformation by exactly solving the full system (4) using LAPACK's* LU *factorization (similar to [TO02, RTSD03]) is significantly slower than computing an approximate but visually equivalent solution by using the incremental QR solver (*IQR*). The number of required basis functions is given in brackets; the LU solver uses all of them. After precomputing deformation basis functions (*Basis Precomp*), shape editing can be done at interactive rates on the* CPU*, and even one further order of magnitude faster on the* GPU.

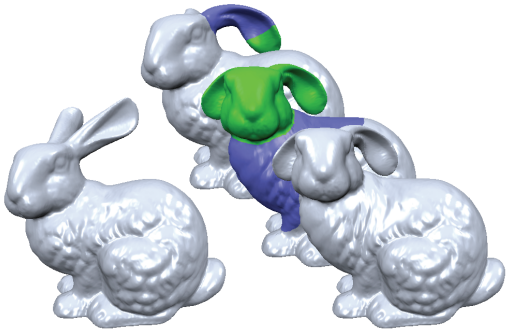

Figure 6: *In three subsequent deformations, the Bunny's ears were bent and its head was lifted and rotated.*

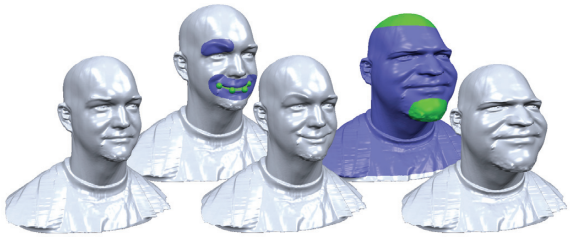

Figure 7: *Deformation of the Male model: Fine-scale editing by applying control curve deformations to mouth and eyebrow (*center*). Coarse scale modification using two simultaneous handles (*right*).*

Model	Δ: [YZX*04]	Δ^2: [BK04a]	RBF
Bunny	5.05 / 0.37	16.4 / 0.254	4.9 / 0.018
Male	11.51 / 0.91	39.2 / 0.271	7.5 / 0.011
Warrior	16.29 / 1.12	58.6 / 0.537	7.6 / 0.032

Table 2: *Comparison of precomputation and per-frame computation cost (in seconds) for different modeling approaches. The timings for Poisson editing [YZX*04] are a very conservative lower bound, since they correspond to only the factorization and back-substitution of a Poisson system, using a more efficient direct solver [BBK05] compared to the original paper. The approach of [BK04a] was used as freeform deformation only and solves bi-Laplacian systems using the same solver. The RBF approach turned out to be more efficient in terms of both precomputation and per-frame costs.*

Table 2 reveals that our GPU-based space deformation runs faster than recent surface-based methods of [YZX*04] and [BK04a], which have to solve linear systems for the free (blue) vertices. Notice that the Dragon and Bust model cannot be handled by the surface-based methods, since the Dragon contains holes in the support region (see accompanying video) and the Bust would require the solution of a 880k×880k sparse linear system. Since the method of [SCOL*04] solves an even three times larger and less sparse least-squares Laplacian system simultaneously for the x, y, and z components, their approach would fail for most of the complex examples due to 2GB main memory limitation.

Fig. 8 compares our method to surface-based freeform deformation and surface-based multiresolution deformation, both based on [BK04a]. While the result of our freeform space-deformation is slightly better than that of the surface-based freeform deformation, the lack of local-frame detail preservation leads to lower quality compared to true multiresolution techniques [BK04a, YZX*04, SCOL*04]. This can also be observed for the extreme bending deformation shown in the accompanying video.

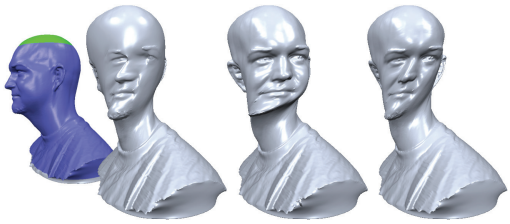

Figure 8: *Comparison of surface-based freeform deformation (*left*), surface-based multiresolution deformation (*center*), and freeform space deformation (*right*).*

The occasionally unintuitive behavior of the space deformation can also be noticed on the spikes of the back of the Dragon's head (cf. Fig. 4), where the technique of [SCOL*04] was shown to yield a more plausible solution. Another limitation of space deformations is that surface parts which have large geodesic distance but small Euclidean distance (e.g., two finger-tips), might influence each other when deformed. However, this problem can usually be resolved by properly restricting the support region and splitting the deformation into two, like it was done for the Bunny's ears (cf. Fig. 6).

Considering the limitations discussed above, the consequent next step for future work is the extension of the presented method to multiresolution modeling. Assuming a multiresolution representation where each point $\mathbf{p}_i \in \mathcal{S}$ is given as a normal displacement of a smooth base surface, i.e., $\mathbf{p}_i = \mathbf{b}_i + \lambda_i \mathbf{n}_i$, the deformed point $\mathbf{p}'_i \in \mathcal{S}'$ can analogously be computed in a vertex shader as

$$\mathbf{p}'_i = \mathbf{d}(\mathbf{b}_i) + \lambda_i \frac{J_{\mathbf{d}}(\mathbf{b}_i)^{-T}\,\mathbf{n}_i}{\left\| J_{\mathbf{d}}(\mathbf{b}_i)^{-T}\,\mathbf{n}_i \right\|}$$

at almost the same speed. However, it is not obvious how to derive the normal or tangent vectors of the displaced vertices on the GPU without requiring transformed local neighborhoods [MBK05].

9. Conclusion

In this paper we presented the necessary techniques to use globally supported radial basis functions for interactive shape editing. Our particular choice of triharmonic RBFs results in deformations of provably optimal fairness, equivalent to surface-based BCM approaches. However, the presented space deformation framework is more general, as it allows the deformation of other explicit surface representations as well.

In order to reduce the otherwise prohibitive computational costs of a naïve implementation, we introduced an incremental least-squares solver, which we used in order to efficiently precompute linear basis functions for the deformation. Evaluating these basis functions on the GPU then allows for real-time shape editing of highly complex models. Each of these three contributions is straightforward to implement and has the potential to reduce the computational complexity by one order of magnitude on its own.

Besides of the presented RBF shape editing, the incremental least-squares solver seems to have many other possible applications. The method of precomputed basis functions can also be used for a wide range of surface- and space-based deformation techniques. Mapping all deformation computations to the GPU is possible for all space-deformation techniques which allow the computation of an (exact or approximate) Jacobian [SP86, HHK92, BR94, KSSH02, LKG*03]. For surface-based deformation approaches it is not clear how to derive an equivalent Jacobian for the normal transformation, therefore the presented GPU-based implementation cannot be transferred directly to surface-based techniques.

Acknowledgments

The authors want to thank David Bommes for help on the implementation and numerous numerical experiments, and Martin Marinov for his triangle reordering algorithm. The Bunny and Dragon models are courtesy of Standford University, the male head is courtesy of Cyberware.

References

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

[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 *Proc. of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 207–217.

[BK03b] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Proc. of Pacific Graphics 03* (2003), pp. 335–343.

[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.

[BR94] BORREL P., RAPPOPORT A.: Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics 13*, 2 (1994), 137–155.

[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.

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

[CRT04] CLARENZ U., RUMPF M., TELEA A.: Finite elements on point based surfaces. In *Proc. of symposium on Point-Based Graphics 04* (2004), pp. 201–211.

[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.

[GL89] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[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.

[Hop99] HOPPE H.: Optimization of mesh locality for transparent vertex caching. In *Proc. of ACM SIGGRAPH 99* (1999), pp. 269–276.

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814.

[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.

[KSSH02] KOJEKINE N., SAVCHENKO V., SENIN M., HAGIWARA I.: Real-time 3D deformations by means of compactly supported radial basis functions. In *Eurographics Short Presentations 02* (2002).

[LKG*03] LLAMAS I., KIM B., GARGUS J., ROSSIGNAC J., SHAW C. D.: Twister: a space-warp operator for the two-handed editing of 3D shapes. In *Proc. of ACM SIGGRAPH 03* (2003), pp. 663–668.

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

[MBK05] MARINOV M., BOTSCH M., KOBBELT L.: Hardware accelerated real-time rendering of multiresolution deformations. Submitted, 2005.

[Mic86] MICCHELLI C. A.: Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation 2* (1986), 11–22.

[MJ96] MACCRACKEN R., JOY K. I.: Free-form deformations with lattices of arbitrary topology. In *Proc. of ACM SIGGRAPH 95* (1996), pp. 181–188.

[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.

[OBS03] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: A multi-scale approach to 3D scattered data interpolation with compactly supported basis fuctions. In *Proc. of Shape Modeling International 03* (2003), pp. 153–161.

[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).

[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.

[RTSD03] REUTER P., TOBOR I., SCHLICK C., DEDIEU S.: Point-based modelling and rendering using radial basis functions. In *Proc. of GRAPHITE 03* (2003), pp. 111–118.

[Sap94] SAPIDIS N. S.: *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*. SIAM, 1994.

[SCOL*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.

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

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

[SPOK95] SAVCHENKO V. V., PASKO A. A., OKUNEV O. G., KUNII T. L.: Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14*, 4 (1995), 181–188.

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (2002), 855–873.

[TRS04] TOBOR I., REUTER P., SCHLICK C.: Reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Proc. of Shape Modeling International 04* (2004), pp. 19–30.

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

[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.

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. of ACM SIGGRAPH 01* (2001), pp. 371–378.