Abstract In this article, we present and discuss three statistical methods for Surface Reconstruction. A typical input to a Surface Reconstruction technique consists of a large set of points that has been sampled from a smooth surface and contains uncertain data in the form of noise and outliers. We first present a method that filters out uncertain and redundant information yielding a more accurate and economical surface representation; the first produces an implicit representation while the second yields a triangle mesh.

Keywords Surface Reconstruction · Point Cloud Denoising · Sparse Implicit · Statistical Learning

1 Introduction

Stochastic methods and concepts are increasingly being found to model natural phenomena better than the hitherto used strictly logical methods, and a “sea change in our perspective” is envisioned when stochastic methods eventually overshadow traditional methods in use and application [45]. Parallels between statistical learning [27] and the workings of the human brain lead mathematicians to believe that such methods could one day help us understand the nature of intelligence itself [55].

The promise and efficacy of these methods is now also being exploited by the Geometric Modeling community for applications including 3D sculpting [75], shape matching [42, 66], best view computation [72], reconstruction of missing data [9], surface fitting [74] and point likelihood estimation [53].

In this paper, we present three methods that apply statistical ideas in the Surface Reconstruction domain. Although such methods are usually slower than their traditional stochastic counterparts, their superior handling of noisy, incomplete and uncertain data makes them especially attractive for Surface Reconstruction. Our first method, presented in Section 2, is a kernel-based approach to scattered data denoising which yields a more accurate and economical representation of the sampled surface.

Next, we present methods to reconstruct a surface from the point set \( \mathcal{P} \) in two standard shape representations, implicit and triangle mesh. In Section 3, we discuss a general framework for the “optimal” selection of the set of centers for implicit basis functions to interpolate \( \mathcal{P} \). The method presented in Section 4 trains a Neural Network to learn the surface represented by \( \mathcal{P} \) as a triangle mesh.

2 Probabilistic Point Cloud Denoising

Point datasets routinely generated by optical and photometric range finders usually contain a small fraction of points with a large error (outliers) and are corrupted by noise. In order to remove these deficiencies from scanned point clouds, a large variety of denoising approaches based on low-pass filtering [39], MLS fitting [1, 20, 41] and partial differential equations (PDEs) [37] has been proposed. These works are motivated by many applications in modeling and rendering [1, 11, 52, 54, 79] which rely on clean data and have become increasingly popular because of the continuous improvement of graphics hardware and technologies for the acquisition of point geometry. While the mentioned denoising approaches remove small-amplitude noise well, they still remain sensitive against outliers.

In this paper, we develop a technique based on non-parametric kernel density estimation [50, 56] to robustly filter a noisy point set which is scattered over a surface and con-
tains outliers. Given 3D scattered points \( \mathcal{P} = \{ \mathbf{p}_1, \ldots, \mathbf{p}_N \} \), we want to estimate an unknown density function \( f(x) \) of the data. A simple density estimation \( \hat{f}(x) \) of \( f(x) \) is for example given by
\[
\hat{f}(x) = \frac{1}{Nh^3} \sum_{i=1}^{N} \Phi \left( \frac{x - \mathbf{p}_i}{h} \right).
\]
The smoothing parameter \( h \) is called the kernel size and \( \Phi \) is the kernel function which is usually chosen to be Gaussian. Figure 1 illustrates the kernel-based density estimation approach. Local maxima of the density estimation \( \hat{f}(x) \) naturally define centers of clusters in the scattered data \( \mathcal{P} \).

The main idea behind our filtering approach consists of defining an appropriate density estimation \( \hat{f}(x) \) to determine those cluster centers which deliver an accurate and smooth approximation of the sampled surface. To detect the local maxima of the constructed density estimation \( \hat{f} \), the Mean Shift technique [15, 17, 23] is used. Clusters corresponding to outliers are then easily detected and can be removed using a simple thresholding scheme.

Recently, robust statistics and statistical learning techniques have gained popularity in Computer Graphics and have been successfully applied to other applications such as data analysis [53] and Surface Reconstruction [21, 30, 63, 65].

We demonstrate that our robust filtering method [60] works nicely on different types of scanned data containing outliers and in combination with different Surface Reconstruction techniques such as Power Crust [2] and Tight Cocone [19].

2.1 Kernel Definition, Convergence and Adaptivity

In this section, we first address the problem of choosing appropriate kernel functions. Our goal is for the resulting density estimation to have local maxima close to the sampled smooth surface. In other words, the density estimation can be interpreted as a likelihood function which reflects the probability that a position in 3D space is located on the sampled smooth surface.

In order to define a likelihood function \( L \), we accumulate local likelihood functions \( L_i \) defined for every sample point \( \mathbf{p}_i \in \mathcal{P} \). We measure the likelihood \( L_i(x) \) for a certain position \( x \) considering the squared distance of \( x \) to the least-squares plane fitted to a spatial neighborhood of \( \mathbf{p}_i \). Being more specific, we determine the fitting plane by computing the weighted covariance matrix
\[
C_i = \sum_{j=1}^{N} (\mathbf{p}_j - \mathbf{c}_i)(\mathbf{p}_j - \mathbf{c}_i)^T \chi \left( \frac{|\mathbf{p}_j - \mathbf{p}_i|}{h} \right),
\]
where \( h \) is the kernel size, \( \chi \) is a monotonically decreasing weight function and \( \mathbf{c}_i \) is the weighted average of all samples inside the kernel. Since \( C_i \) is symmetric and positive semi-definite, its eigenvalues \( \lambda_i^{(1)}, i = 1, 2, 3 \), are real-valued and non-negative: \( 0 \leq \lambda_i^{(1)} \leq \lambda_i^{(2)} \leq \lambda_i^{(3)} \). Furthermore, the corresponding eigenvectors \( \mathbf{v}_i^{(1)} \) form an orthonormal basis. Thus the covariance matrix (1) defines the ellipsoid
\[
E_i(x) = \{ \mathbf{x} : (\mathbf{x} - \mathbf{c}_i)^T C_i^{-1} (\mathbf{x} - \mathbf{c}_i) \leq 1 \},
\]
where the least-squares fitting plane is spanned by the two main principal axes \( \mathbf{v}_i^{(1)} \) and \( \mathbf{v}_i^{(2)} \) of \( E_i \) and has the normal \( \mathbf{v}_i^{(1)} = \mathbf{n}_i \). A 2D example is illustrated in Figure 2.

If normals are provided by the scanning device we use them instead of the estimated normals. Using the squared distance of \( x \) to the least-squares plane, we measure the likelihood \( L_i(x) \) as
\[
L_i(x) = \Phi_i(x-c_i) \left[ h^2 - |(x-c_i) \cdot \mathbf{n}_i|^2 \right].
\]

Thus, positions \( x \) closer to the least-squares plane will be assigned a higher probability than positions that are more distant. Additionally, we assume that the influence of a point \( \mathbf{p}_i \) on the likelihood of a position \( x \) diminishes with increasing distance. To consider this behavior, we use monotonically
Fig. 3 A slice of the likelihood function $L$ of the noisy Buddha model (left) and zooms of the framed regions (right). The function values are represented by colors increasing from blue to purple. Note that $L$ is a smooth function.

decreasing weighting functions $\Phi_j$ to reduce the influence of each $L_i$. In contrast to radial functions in [49, 53], we use a trivariate anisotropic Gaussian function $\Phi_j$ which is adapted to the shape of the ellipsoid $E_j$. This has the advantage that the weighting function is also adapted to the point distribution in a spatial neighborhood of $p_i$.

To define the likelihood function $L$ modeling the probability that a certain point $x$ lies on the sampled surface $S$, we accumulate the local likelihoods $L_i(x)$ contributed by all points $p_i$.

$$L(x) = \sum_{i=1}^{N} w_i L_i(x)$$

Note that we can easily incorporate scanning confidence measures $w_i \in [0, 1]$ associated with each point $p_i$ by scaling the amplitudes of the likelihood functions. If no scanning confidences are provided, we use $w_i = 1$. Figure 3 shows an example of a slice of the likelihood function $L$.

After determining the likelihood function $L$, we use it to smooth the point cloud by moving all samples to positions of high probability. This means we move the samples to positions which are most likely locations on the sampled surface. To find the local maxima of $L$, we use a procedure similar to a gradient-ascent maximization. We freeze the weighting functions $\Phi_j$ since they change slowly and approximate $\nabla L(x)$ by

$$-2 \sum_{j=1}^{N} w_j \Phi_j(x - c_j) \cdot \left( (x - c_j) \cdot n_j \right) \cdot n_j.$$

$$\tau = \frac{1}{2 \sum_{j=1}^{N} w_j \Phi_j(x - c_j)}.$$  (3)

This means that the step size is small near the probability maximum and increases towards the border of each kernel. This provides a fast and stable convergence of all sample points.

Combining equations (2) and (3), we get the resulting iterative scheme

$$p_i^{k+1} = p_i^k - m_i^k$$

$$m_i^k = \frac{\sum_{j=1}^{N} w_j \Phi_j(p_i^k - c_j) \cdot \left( (p_i^k - c_j) \cdot n_j \right) \cdot n_j}{\sum_{j=1}^{N} w_j \Phi_j(p_i^k - c_j)}.$$  (4)

In order to filter the point cloud $P$, we apply the iterative scheme individually to every sample. We stop the iterative process when

$$||p_i^{k+1} - p_i^k|| < 10^{-4} h.$$  (5)

Each sample usually converges in less than 10 iterations.

A feature of our filtering method is the inherent clustering property. As the number of kernels is larger than the number of maxima in the likelihood function $L$ (see Figure 1), several sample points converge to the same probability maximum. We cluster those samples and place one representative point at the local maximum of $L$. See Table 1 for details on the point reduction rate.
Fig. 5 Filtering of a face scan acquired using a structured light scanner. Initial scattered point data contains scanning artifacts and outliers (left). Our method automatically removes the outliers and nicely suppresses the defects (right).

So far we only used a fixed radius $h$ to compute the local neighborhoods for the ellipsoidal weight function and least-squares fitting plane computation. However, invariant kernels might not be suitable for datasets with varying sampling density. To overcome this problem, we use the $k$-neighborhood of each sample $p_i$ for the PCA analysis to compute the ellipsoidal kernel $E_i$. In this manner, we not only adapt the kernel shape to the point sample distribution in a neighborhood of $p_i$ but also the kernel size to the spatial sampling density. The motivation behind this choice can be observed in Figure 4. If a fixed radius $h$ is used, local maxima of $L$ are created distant to the most likely surface in regions of the point cloud with large-amplitude noise. Those maxima also attract points during the iterative filtering process creating a second layer of points around the most likely surface (left image). The usage of adaptive kernels leads to larger kernel sizes in these regions due to the lower sampling density of large-amplitude noise. Therefore, kernels of both layers intersect which dampens the effect of local maxima. This results in an improved filtering of large scale noise (right image).

2.2 Discussion

Results of our denoising approach on structured light scans and laser scanned data show that it has a good performance on different types of scanned data. Experiments illustrate the strength of our method in removing outliers, especially 3D “salt and pepper” outliers. Finally, we see that results of well-known Surface Reconstruction methods can be improved in conjunction with our filtering method. Table 1 summarizes timings and the parameters used to generate the results.

In Figure 5, we show a point cloud face dataset acquired by a structured light scanner before and after filtering using our method. The raw point cloud suffers from several outliers and ridges which are typical artifacts caused by the structured light. We show this comparison to illustrate the effectiveness of our method in removing outliers and for smoothing difficult datasets. Due to the clustering property of our method, groups of outliers usually converge to a set of single points sparsely distributed around the surface samples. These points can be characterized by a very low spatial sampling density compared to the surface samples. We use this criterion to detect outliers and remove them using simple thresholding. Figure 7 shows an additional example with a large amount of randomly generated points which can be interpreted as 3D “salt and pepper” outliers. In the case of images, “salt and pepper” noise corrupts random image pixels with intensity spikes. This means that a number of pixels in the image have a very large intensity difference to neighboring pixels. For point clouds, we can model this kind of noise by displacing points of the dataset far from the smooth surface. In our example, we move points inside the bounding box of the dataset. Additionally, we add noise to the normals by perturbing them with random angles. Although the outlier density is high as shown in Figure 7, our algorithm is able to remove the noise and the outliers properly.

In Figure 6, we demonstrate the filtering efficiency of our algorithm on laser scanned data. We show this comparison as laser scans are usually affected by different types of noise compared to structured light scans. Due to the different acquisition technique, laser scans are usually not corrupted by ridges and pits caused by structured light. Instead, they are affected by dense small-amplitude noise. Figure 6 illustrates that high-frequency noise is removed by our method while lower frequency details like hair, mouth and eyes of the Bimba model are preserved. As noted previously, our method uses adaptive kernels to handle large scale noise. Figure 4 shows that while the dragon scan cannot be filtered accurately using a fixed kernel size, adaptive kernels provide a proper filtering of large amplitude noise.

An interesting application of our denoising method is to preprocess noisy data before a surface is reconstructed. Usu-
Fig. 8 Figures (a) and (b) present the head of the Dragon scans from the Stanford Scanning Repository before and after our filtering procedure. Figures (c) and (d) show zooms of the images (a) and (b) close to the tongue region. Notice that noise is removed and that the filtered samples indicate a surface. Figures (e) and (g) illustrate Power Crust and Tight Cocone reconstructions from the noisy samples shown in (a). Figures (f) and (h) show reconstruction results from the filtered data shown in (b). While the Power Crust algorithm shows noticeably improved results with small defects (f), the Tight Cocone algorithm reconstructs a smooth mesh (h).
Table 1: Timings for the ellipsoid kernel computation and the filtering for the models presented in this paper. The kernel size $h$ is chosen in the interval of one to ten times the average sampling density of the input data. The character $N$ denotes the number of input samples and $M$ the number of filtered points. The parameter $k$ indicates the number of nearest neighbors used for the adaptive kernel computation. All results were computed on a 2.66 GHz Pentium 4 with 1.5 GB of RAM.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$N$</th>
<th>$M$</th>
<th>kernels</th>
<th>filtering</th>
<th>$h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>180K</td>
<td>114K</td>
<td>1.38s</td>
<td>18.45s</td>
<td>0.3</td>
</tr>
<tr>
<td>Bunny</td>
<td>362K+25K</td>
<td>324K</td>
<td>3.2s</td>
<td>52s</td>
<td>0.001</td>
</tr>
<tr>
<td>Bimba</td>
<td>1.9M</td>
<td>1.2M</td>
<td>16s</td>
<td>80s</td>
<td>1</td>
</tr>
<tr>
<td>Dragon head</td>
<td>485K</td>
<td>170K</td>
<td>23.22s</td>
<td>10m 53s</td>
<td>0.0015</td>
</tr>
<tr>
<td>Dragon</td>
<td>2.1M</td>
<td>796K</td>
<td>1m 43s</td>
<td>36m 26s</td>
<td>0.0015</td>
</tr>
<tr>
<td>Dragon</td>
<td>2.1M</td>
<td>795K</td>
<td>6m 40s</td>
<td>38m 05s</td>
<td>25K k</td>
</tr>
</tbody>
</table>

Fig. 7: Left: Raw registered range scans of the Stanford Bunny dataset expanded by 25K random “salt and pepper” outliers. Right: Our denoising approach. For surface reconstruction, we use the number of filtered points. The parameter $k$ denotes the number of input samples and $M$ the number of filtered points. The parameter $k$ indicates the number of nearest neighbors used for the adaptive kernel computation. All results were computed on a 2.66 GHz Pentium 4 with 1.5 GB of RAM.

3 Surface Reconstruction with Sparse Implicits

In implicit modeling [10], a 3D data set $\mathcal{P} := \{p_i : i = 1, \ldots, N\}$ is approximated by an implicit surface $\Sigma := \{ x \in \mathbb{R}^3 : f(x) = 0 \}$, and the function $f(x) := \sum_{i=1}^{N} \alpha_i \phi_i(x)$ is a linear combination of the basis elements $B := \{ \phi_i(x) \}_{i=1}^{N}$. The underlying mathematical framework builds on numerical linear algebra and the degrees of freedom on the choice of $B$ (e.g. globally [13, 70] and compactly [43, 47] supported RBFs, Partition of Unity [46, 76], Moving-Least-Squares methods [1, 22, 35, 38, 52, 64]) enable to adapt the model parameters to specific problem constraints such as huge data sets with attributes, local accuracy, and degree of smoothness. Furthermore, multi-resolution techniques have been recently proposed [68] and a wide range of applications, including deformation, fast rendering, and collision detection [3, 48, 69], have been targeted by several authors.

In our formulation, we consider a Reproducing Kernel Hilbert Space $\mathcal{H}$ (RKHS, for short) [4] with kernel $\Phi(x, y)^\dagger$; in this case, each basis function $\phi_i(x) := \Phi(x, p_i), i = 1, \ldots, N$, is centered at a point of the input data set. Among the properties of RKHS, we remind the reproduction property

$$h(x) := <\Phi(x, y), \Phi(x, y) >_{\mathcal{H}}, \forall h \in \mathcal{H}, \forall x, y \in \mathbb{R}^d, \quad (4)$$

that will be used in the following discussion.

A sparse approximation method searches, among all the possible approximations of $f$ with the same error, for the one $f^\star$ that involves the smallest number of basis functions. In terms of the corresponding iso-surfaces, this is equivalent to approximating $\Sigma$ with $\Sigma^\star := \{ x \in \mathbb{R}^3 : f^\star(x) = 0 \}$.

Previous work on surface sparsification can be subdivided into the following groups: local, global, and clustering techniques. Local methods build a smooth surface through an iterative and multi-scale procedure based on a local polynomial approximation. In this case, the centers and radii are determined by a posteriori updates of the model and guided by the local approximation error [13, 14, 34, 48, 64]. Global methods find a sparse representation by minimizing a constrained convex quadratic optimization problem [24, 51, 65, 74]; a detailed discussion on them will be given shortly in this section. Since each $\phi_i$ is centered at a point $p_i$ of $\mathcal{P}$, clustering techniques can also be used to select the centers of the sparse representation. The idea is to group those points which satisfy a common “property” and to center a basis function at a representative point of each cluster. Planarity and closeness, measured in the Euclidean space using the polynomial of degree $d$.

$$\Phi(x, y)^\dagger \text{ or the polynomial of degree } d \Phi(x, y)^\dagger := (1 - <x, y>)^d.$$
On Stochastic Methods for Surface Reconstruction

Fig. 10 (a) Input Bernoulli lemniscate with $N$ centers depicted by black dots, (b-d) curve approximations with different percentages of selected centers: $n$ is the number of iterations in (8).

Fig. 9 (a) Input points on a Bernoulli lemniscate and initial set of centers (yellow circles), (b-c) first and last iteration of the $k$-means clustering [40], (d) reconstructed curve and iso-contours of the associated scalar field.

Following [24], the quality of the approximation of $f$ with $f^*$ is measured by the quadratic misfit error $\|f - f^*\|_{p^*}^2$ and the selection of the basis functions which contribute to $f^*(x) := \sum_{i=1}^N a_i \Phi(x)$ is given by the coefficients $a_i \neq 0$, $i = 1, \ldots, N$. The sparsification value, i.e. the number of basis functions used in $f^*$, is quantified by the $l_1$-norm $\|a\|_{l_1} := \sum_{i=1}^N |a_i|$ of the vector $a := (a_i)_{i=1}^N$. Then, we consider a compromise between these two terms and minimize the functional

$$F(a) := \frac{1}{2} \|f - \sum_{i=1}^N a_i \Phi(x, p_i) \|_{p^*}^2 + \epsilon \|a\|_{l_1}$$

where $\epsilon (> 0)$ is the tradeoff between the misfit measure and sparsity. If $\epsilon = 0$, we get the standard least-squares approximation scheme, while by increasing $\epsilon$ we neglect a greater number of basis functions and accept a lower approximation accuracy.

As shown in [24], replacing each unknown $a_i$ of (5) with a pair of positive variables $(a_i^+, a_i^-)$, such that $a_i = a_i^+ - a_i^-$, results in the following constrained convex quadratic optimization problem (i.e. Support Vector Machine) [24]:

$$\min_{a_i^+, a_i^-} \{ F(a^+, a^-) \}$$

with

$$F(a^+, a^-) := \frac{1}{2} \sum_{i,j=1}^N \Phi(p_i, p_j)(a_i^+ - a_i^-)(a_j^+ - a_j^-) - \sum_{i=1}^N y_i (a_i^+ - a_i^-) + \epsilon \sum_{i=1}^N (a_i^+ + a_i^-),$$

$$f(p_i) = y_i, \ i = 1, \ldots, N,$

and feasible set

$$\mathcal{F} := \{ a^+ \geq 0, a^- \geq 0, \sum_{i=1}^N (a_i^+ - a_i^-) = 0, \ a_i^+ a_i^- = 0, \ i = 1, \ldots, N \}.$$ 

This last formulation is equivalent to (5) and facilitates its numerical optimization by removing the absolute values. If $\mathcal{F}$ is a large data set, defining a sparse representation of $f$ as a solution of the quadratic minimization problem (6) with $2N$ unknowns is generally infeasible due to the amount of input data, the possible ill-conditioning of the coefficient matrix $L := \{ L_{ij} \}_{i,j=1,\ldots,N}$, $L_{ij} := \Phi(p_i, p_j)$, and the unbounded feasible set $\mathcal{F}$. Therefore, all these factors badly affect the stability and convergence of iterative methods [16, 25]. In [63, 65], (6) is solved by applying a decomposition method and a heuristic coordinate descendent optimization scheme respectively. In the last case, at each iteration $(2N - 1)$ variables are fixed and the objective function is minimized with respect to the remaining free parameter.

To avoid the formulation in (6), which is a consequence of the $C^0$-regularity of the $l_1$-norm in (5), we use the smooth approximation

$$\|a\|_{l_1} \approx \sum_{i=1}^N |a_i|^2 + \eta^{1/2}, \ \ \ \eta \to 0,$$
and we replace the functional (5) with

$$G(a) := \frac{1}{2} \| f - \sum_{i=1}^{N} \Phi(x, p_i) \|_{\mathcal{H}}^2 + \epsilon \sum_{i=1}^{N} [a_i^2 + \eta]^{1/2}.$$ 

In the following, we prove that minimizing $G$ is equivalent to solving a system of non-linear equations. Using the reproduction property (4), we can rewrite $G$ as

$$G(a) \equiv \frac{1}{2} \sum_{i,j=1}^{N} \Phi(p_i, p_j) a_i a_j - \sum_{i=1}^{N} y_i a_i + \epsilon \sum_{i=1}^{N} [a_i^2 + \eta]^{1/2} + \frac{1}{2} \| f \|_{\mathcal{H}}^2,$$

where the term $\frac{1}{2} \| f \|_{\mathcal{H}}^2$ is constant. Then, the critical points of $G$ are the solutions of the following system of non-linear equations

$$\nabla G = 0 \iff \begin{bmatrix} L + \epsilon \Delta(a(n)) \end{bmatrix} a^{(n+1)} = y \iff a^{(n+1)} = \left( L + \epsilon \Delta(a(n)) \right)^{-1} y,$$

minimum. The solution of (7) is evaluated by running the iterative scheme

$$[L + \epsilon \Delta(a^{(n)})] a^{(n+1)} = y \iff a^{(n+1)} = [L + \epsilon \Delta(a^{(n)})]^{-1} y,$$

with $a^{(0)}$ initial guess and $n \geq 1$. The term $a^{(n+1)}$ is achieved from $a^{(n)}$ by solving a linear system with direct or iterative solvers, e.g., the Gauss-Seidel or conjugate gradient method [25]. The iterative procedure stops when the solution becomes stationary, i.e., we do not improve the number of null coefficients and/or the residual error between two consecutive iterations is below a given threshold.

3.1 Discussion

As done in [70], the initial set of centers is given by $\mathcal{P}$ plus $2N$ additional centers $\{p_i \pm \delta n_i\}_{i=1}^{N}$ placed at a small distance $\delta$ from $p_i$, where $n_i$ is an approximation of the normal at $p_i$. Recently [74], the surface normal vectors have been incorporated in the regularization framework, thus avoiding off-set conditions to guarantee a non-null solution. This approach uses curvature and grid-based clustering on point clouds to guide the selection of the basis function centers and radii, and to achieve high-quality approximations.

Figures 10 and 11 show the curve reconstruction with several sparsification percentages ($\Phi$ is the Gaussian Kernel); we note that the local noise and irregular sampling, which affect the approximation in Figure 11(a), are attenuated in (b) where we use a minor number of basis functions. This property is due to the misfit error in (5) and to a lower conditioning number of the coefficient matrix related to the least-squares approximation. Furthermore, a smooth kernel function $\Phi$ and the induced norm $\| \cdot \|_{\mathcal{H}}$ provide a smooth solution, thus reducing the influence of noise and outliers of $\mathcal{P}$ in the reconstructed surface. Finally, Figure 11(c-d) shows the center selection on a curve with a non-uniform sampling.

Using a $k$-neighborhood of each center [5] requires a storage overhead of $(3k+1)/N$ non-null entries for the matrix $L$. Then, each iteration (8) updates only the principal diagonal of $L$, preserves its sparsity and positive definiteness, and requires $O(N)$ time to solve the linear system. The proposed sparsification scheme is equivalent to Support Vector Machine and it avoids the constrained convex quadratic minimization and the use of heuristics involved in SVMs. Since the sparsification starts from the full resolution with $3N$ basis functions, we get a fine-to-coarse approach and the iterative procedure (8) builds a multi-level approximation scheme based on a sequence of nested spaces; for more details, we refer the reader to [51]. If we get a complete sparsification, i.e., the iterative solver of the system of non-linear equations converges to the null solution, each approximation is achieved by using the intermediate iterations (see Figure 12).
As the iterations in (8) proceed, the $L_\infty$ error between the current approximation $f^{(n)}(x) = \sum_{i=1}^{N} u_i^{(n)} \varphi_i(x)$ and $f$, measured by

$$L_\infty(f, f^{(n)}) := \max_{i=1,\ldots,N} \{|f(p_i) - f^{(n)}(p_i)|\},$$

increases due to a minor number of basis functions. Unlike local approximation techniques, which are capable of adapting the center selection to local accuracy through an extensive use of function evaluations, we cannot use this error as a stop criterion due to the global formulation of our problem.

The fine-to-coarse structure requires a storage overhead and computational cost greater than local approximation [1, 47, 48]. For instance, if $k = 20$ we are able to handle at least 500K centers on a Pentium IV 2.80 GHz with 1 GB RAM; in this case, the evaluation of $L_1$ and the sparsification scheme require approximately 90 to 180 seconds. Our method always converges to a global minimum and it can be used for those applications where center selection is mandatory to speed up surface sampling, interactive modeling and queries [12, 65]. However, it cannot deal with huge data sets, which can be handled efficiently by [46, 48, 74].

Setting the support of the basis functions is a delicate part of the approximation and sparsification scheme; in fact, its choice affects the size of the details that will be recovered as well as the maximum sparsification percentage, which avoids artifacts in the reconstructed model. In our framework, the support associated to the basis function $\varphi_i$ is set equal to the minimum radius of the sphere centered at $p_i$ that contains the $k$-nearest neighbors of $p_i$, and $k$ varies from 10 to 20 depending on the number of input points. Our tests have shown that these values lead to a good compromise between sparsification rate and approximation accuracy (see Figure 13). Finally, in the examples of Figures 12 and 13 we used $\Phi(r) := (1-r)^4 (4 + 16r + 12r^2 + 3r^3)$ [43, 62] as the sparse kernel, where $r = \frac{\|x-y\|}{\sigma}$ and $\sigma$ is its compact support.

4 Shape Learning from Point Clouds

In this section, we present Neural Meshes, a technique to learn the shape of a 3D data set $\mathcal{P} := \{p_i(x_i, y_i, z_i) : i = 1, \ldots, N\}$ by training a Neural Network [8]. Previously, Neural Networks have been trained to reconstruct parametric and freeform surfaces [26, 44, 77] representing $\mathcal{P}$, whereby the Neural Network learns a function $f(x, y)$ such that $|f(x_i, y_i) - z_i| < \epsilon$, $\forall i = 1 \ldots n$ and for some acceptable error level $\epsilon$. An extension of Neural Networks, functional networks, have been used to reconstruct $\mathcal{P}$ with B-spline and Bezier surfaces [29] (see also references therein).

The above methods fall under the supervised learning category; they assume a relationship between the input variables. Unsupervised learning methods make no such assumption. The Neural Mesh technique also belongs to this class of methods. In this class, the Neural Network is not trained to compute a surface. Instead, the trained Neural Network is the desired surface. The methods described in [6, 7, 28, 71] learn control grids for reconstruction of $\mathcal{P}$ and parametric grids for subsequent parameterization of $\mathcal{P}$. Neural Networks can also be trained to directly interpolate or approximate $\mathcal{P}$ [36, 78].

In the unsupervised learning methods described above, the topology and number of vertices of the learnt surface remain unchanged since initialization. As they initialize the surface with a 2D grid, they can accurately represent $\mathcal{P}$ only if $\mathcal{P}$ represents a surface patch. Also, the learnt surface may under-represent detailed features in $\mathcal{P}$. As a solution to the latter problem, subdivision is suggested in [78]. In [6, 71], where the surface has the topology of a quad grid, the authors suggest tracking the activity of each vertex with an associated counter, which increases each time the vertex participates in learning. They can then spot active vertices by their high counter values, and add entire rows/columns...
of vertices in their neighborhoods. Both these solutions are global in nature and end up adding new vertices in unwanted regions of the surface as well.

A Neural Mesh is initialized as a closed triangle mesh, \( \mathcal{M} \). Each vertex in \( \mathcal{M} \) stores a counter value, \( \tau \), and a winning sample number, \( S_v \). \( \mathcal{M} \) learns from each training sample, \( s \), from \( \mathcal{P} \) by moving the corresponding winner vertex, \( v_w \), towards the sample and applying smoothing to its 1-ring neighbors. An illustration for the 1D case is given in Figure 14a. The winner’s new position is given as \( v_w \leftarrow v_w + \alpha_d \cdot F(d) \cdot d \), where \( d = \frac{1}{v_w} \delta \) and \( \alpha_d \) is a parameter between 0 and 1. \( F(d) \) filters out the effects of outliers in \( \mathcal{P} \).

A moving average, \( \mu_t \), is calculated as \( \mu_t = \frac{1}{\tau} \sum_{\epsilon \leq d < \epsilon + 1} \mu_{t-1} \). A threshold is then calculated as \( \epsilon_d = \mu_t + \alpha_d \sigma_d \), using an input tolerance \( \alpha_d \), and \( F(d) \) is defined as

\[
F(d) = \begin{cases} 
1 & \text{if } |d| \leq \epsilon_d \\
\frac{1}{d} & \text{if } |d| > \epsilon_d 
\end{cases}
\]

Smoothing the winner’s neighborhood avoids foldovers and local minima in \( \mathcal{M} \), illustrated for the 1D case in Figure 15. For each vertex, \( v_i \), in the 1-ring of \( v_w \), its Laplacian [67] is calculated, \( L(v_i) = \frac{1}{\sigma_i} \sum_{\epsilon} (v_k - v_i) \), followed by the displacement, \( L_x(v_i) = L(v_i) - (L(v_i) \cdot n) n \), where \( n(v_i) \) is \( v_i \)'s valence, \( v_k \)'s 1-ring neighbors and \( n \) its approximated normal. \( v_i \)'s position is then updated, \( v_i \leftarrow v_i + \alpha_x L_x(v_i) \), where \( \alpha_x \) is a smoothing parameter between 0 and 1.

The winning vertex’s counter, \( \tau(v_w) \), is then updated using \( v_w \)'s winning sample number, \( S_w(v_w) \), and the current sample number, \( S_c \). First, the number of samples since the current winner’s last win are computed, \( x = S_c - S_w(v_w) \). Then the updates are made, \( \tau(v_w) \leftarrow \alpha_{\tau\tau} \tau(v_w) + 1 \), and \( S_w(v_w) \leftarrow S_c \). \( \alpha_{\tau\tau} \) is calculated as \( \alpha_{\tau\tau} = \left( \frac{1}{2} \right) \frac{1}{X} \), where \( X \) is the current number of vertices in \( \mathcal{M} \) and \( \lambda \) is an input parameter such that a vertex loses half its counter value if it is not the winner for \( \lambda N \) samples.

As intended, active vertices are now identified by their high counter values. Every \( C_{\text{add}} \) samples, all counter values are synchronized, where \( C_{\text{add}} \) is an input parameter. For each vertex, \( v_i \), in \( \mathcal{M} \), the number of previous non-winning samples is calculated, \( x = S_c - S_w(v_i) \), and the updates are made, \( \tau(v_i) \leftarrow \alpha_{\tau\tau} \tau(v_i) \), and \( S_w(v_i) \leftarrow S_c \). Then a vertex split operation is performed on the vertex with the highest counter value. Counter synchronization is also performed after every \( C_{\text{rem}} \) samples, where \( C_{\text{rem}} \) is input by the user. This time, vertices with counter values lower than a certain threshold are removed from \( \mathcal{M} \) using half-edge collapse operations. Addition/removal of vertices is illustrated in Figure 14b, which also shows the vertices whose valences are affected by the operations. Half-edge collapse operations that would cause \( \mathcal{M} \) to become non-manifold (Figure 16a,b) are not performed. For more details, we refer the reader to [31, 58].
Operations affecting the topology of $M$ are invoked after every $N\text{C}_{\text{top}}$ samples, where $C_{\text{top}}$ is chosen by the user and $N$ is the current number of vertices in $M$. The average triangle area, $\overline{A}$, in $M$ is used to calculate a triangle removal and a boundary merging threshold. Triangles with area greater than the triangle removal threshold are removed, and boundaries whose Hausdorff distance to each other is less than the boundary merging threshold are merged. If the removal of a triangle causes $M$ to no longer be manifold, neighboring triangles are removed to restore manifoldness (Figure 16c). Figure 17 shows the effect of these steps.

4.1 Discussion

Neural Meshes effectively solve the problems unaddressed by previous surface learning methods. They can represent entire surfaces, not just patches. Learning is adaptive; it starts with a small, simple initial surface to which vertices are added only where needed, i.e. in the 1-ring neighborhood of active vertices. Vertices that become misplaced during training and over represent $P$ are removed. Also, neural meshes possess the ability to learn topology.

Notice that the only step of the algorithm where $P$ is required is in picking training samples, thus making the running time independent of the size of $P$. This is in direct contrast to methods that need to process all input points in order to output a surface. Independence from the input point cloud also allows out-of-core processing of large data sets.

The coarse-to-fine way of shape learning in Neural Meshes offers a deeper insight into the shape. Vertices in $M$ in high curvature areas display large variations in their normals during training compared to those representing flatter areas. This information can be tracked [32] by the vertices’ counters, leading to higher counter values, and thus higher vertex population, in high curvature areas. Notice that in comparison, the default Neural Meshes mimic the density of $P$. Some reconstructions are shown in Figure 18.

Like most learning algorithms, Neural Meshes suffer from the need for user parameters. While a default set of values can be set, best results will be obtained by tuning the parameters in accordance to the input set. This could be seen as an advantage for the expert user. For an extensive treatment of this issue, we refer the reader to [58].

Neural Meshes also inherit long running times from learning methods. The complexity of the algorithm described above is $O(N^2)$, based mainly on sorting vertices according to their counter values to find the ones with highest and lowest values. In an alternate implementation of the algorithm, the vertices are copied to a priority queue data structure, where a vertex’s counter value is replaced by its position in the priority queue. It has been shown [58] that implementing the priority queue as a self-balancing binary tree reduces the complexity of the Neural Mesh algorithm to $O(N \log N)$ with no significant difference to output mesh quality.

Despite the speedup offered by the priority queue implementation, the method is slow and non-competitive with contemporary geometry-based Surface Reconstruction methods. The majority of the running time is spent in Geometry Learning. We expect that a shrink-wrapping approach with the Neural Mesh initialized as an inflated bounding sphere
with number of vertices close to the final number could offer a solution to this problem.

5 Conclusion

In this paper, we presented three stochastic methods for Surface Reconstruction. Despite the fact that some of these methods are relatively slow, we believe them to be important as they represent the entry of stochasticity into Geometric Modeling. We believe that in their natural ability to reliably deal with uncertain and fuzzy data, these methods hold the key to many problems in Geometric Modeling where one typically works with measurements with exactly these defects.

Acknowledgements

We would like to thank Tamal Dey and Nina Amenta for making their surface reconstruction software available. The Dancer and Bimba models are courtesy of the AIM@SHAPE Shape Repository and the Buddha, Dragon, and Bunny datasets are courtesy of the Stanford 3D Scanning Repository. This research is partially supported by the European FP6 NoE grant 506766, AIM@SHAPE.

References

Waqar Saleem is a PhD student at the Max-Planck-Institut (MPI) Informatik, where he works on statistical methods for shape applications, automatic view generation of 3D models and shape similarity. He is also involved in maintenance and development of the AIM@SHAPE Shape Repository. He has completed his MSc (2004) and BSc (2002) degrees in Computer Science resp. from Saarland University and Mohammad Ali Jinnah University, Karachi.

Oliver Schall received a Diploma in Computer Science from RWTH Aachen University in 2003 and is now a Ph.D. student at the Max-Planck-Institut (MPI) Informatik in Saarbrücken, Germany. His current research interests include denoising of static and time-varying geometric data and surface reconstruction.

Giuseppe Patané received a Ph.D. in “Mathematics and Applications” from the University of Genova (2005) and a Post-Laurea Degree Master in “Application of Mathematics to Industry” from the “F. Severi National Institute for Advanced Mathematics” - University of Milano (2000). Currently, he is a Research Fellow in the Shape Modelling Group at IMATI-CNR, Genova, Italy. His research interests include numerical linear algebra, surface approximation, parameterization, and analysis.

Alexander Belyaev is currently a Senior Researcher at the Computer Graphics Department of the Max-Planck-Institut (MPI) Informatik, Saarbrücken, Germany. His research interests include geometric modeling and processing and homogenization of PDE operators. Belyaev received his M.S. and Ph.D. degrees in Mathematics from Moscow State University in 1986 and 1990, respectively.

Hans-Peter Seidel is the scientific director and chair of the computer graphics group at the Max-Planck-Institut (MPI) Informatik and a professor of computer science at the University of Saarbruecken, Germany. Seidel has published some 200 technical papers in the field and has lectured widely on these topics. He has received grants from a wide range of organizations, including the German National Science Foundation (DFG), the German Federal Government (BMFBBF), the European Community (EU), NATO, and the German-Israel Foundation (GIF). In 2003 Seidel was awarded the ‘Leibniz Preis’, the most prestigious German research award, from the German Research Foundation (DFG). Seidel is the first computer graphics researcher to receive this award. In 2004 he was selected as founding chair of the Eurographics Awards Programme.