Filling Holes in Point Clouds

Pavel Chalmovianský[1], Bert Jüttler[2]

[1] Johannes Kepler University, Spezialforschungsbereich SFB 013
Freistädter Str. 313, 4040 Linz,
Pavel.Chalmoviansky@jku.at,
http://fractal.dam.fmph.uniba.sk/~chalmo
[2] Johannes Kepler University, Dept. of Applied Geometry,
Altenberger Str. 69, 4040 Linz,
bert.juettler@jku.at,
http://www.ag.jku.at

Abstract. Laser scans of real objects produce data sets (point clouds) which may have holes, due to problems with visibility or with the optical properties of the surface. We describe a method for detecting and filling these holes. After detecting the boundary of the hole, we fit an algebraic surface patch to the neighbourhood and sample auxiliary points. The method is able to reproduce technically important surfaces, such as planes, cylinders, and spheres. Moreover, since it avoids the parameterization problem for scattered data fitting by parametric surfaces, it can be applied to holes with complicated topology.

Keywords. Reverse engineering, scattered data, algebraic surface fitting, meshless methods

1 Introduction

Since the advent of advanced laser scanners, even complicated objects can be digitized with impressive accuracy. This led to the technology of reverse engineering [18], which has been developed into a valuable alternative to the traditional top–down construction process in CAD. Instead of designing a CAD model from scratch, the model is (semi–) automatically created from the cloud of measurement data.

The data acquired by the scanning device, however, may have various problems. For instance, some parts of the objects can be missing, due to problems with accessibility/visibility, or due to the special physical properties of the scanned surface (e.g. transparentness, reflectivity, etc.). This produces holes in the data set, which do not correspond to any holes in the object (cf. [5, 17]).

There are several possible ways to address this problem. For instance, one may try to combine several views, i.e., several scanned point sets of the same object. As another approach, the holes can be filled with auxiliary, artificially generated points. The latter approach will work also in regions where the object is difficult to scan, due to visibility and/or optical properties. Also, it can be used even if the problems have not been realized immediately during the scanning process.

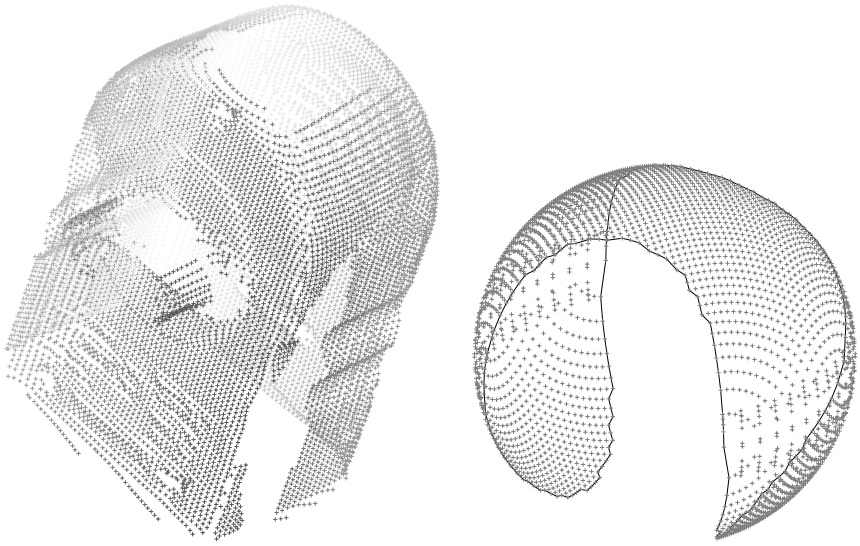

Fig. 1. Left: babyphone data, generated by a 3D Laser scanner. Right: tennis ball data (artificially generated).

In this paper, we use algebraic surfaces for filling the hole by generating auxiliary points. After detecting the hole, a surface is fitted to the neighbourhood of its boundary. The method is able to reproduce spheres, planes and circular cylinders. Finally, by sampling points from the part of the algebraic surface which corresponds to the hole, the missing points are constructed.

This paper is organized as follows. The next section describes basic notions and facts used throughout the paper. Section 3 summarizes the algorithms for estimating normals associated with the points. Techniques for detecting boundary points and for constructing polygonal boundaries for the holes are outlined in section 4. Section 5 describes the algebraic surface fitting. The sampling process used for generating the new points is introduced in section 6. Finally, we conclude this paper. Technical details can be found in the appendices.

2 Preliminaries

The data generated by the scanner form a set $P = \{\mathbf{p}_0, \ldots, \mathbf{p}_N\} \subset \mathbb{R}^3$, where $N \in \mathbb{Z}_+$ is the number of points. Typically, N is in the order of tens of thousands for objects of a size of coffee mug, see Figure 1.

For the convenience of the reader, we summarize some facts about the Bernstein–Bézier representation of trivariate polynomials (see [6] for more details).

Let $\mathbf{v}_i \in \mathbb{R}^3$ for $i \in \{0, 1, 2, 3\}$ be four non-coplanar points with coordinates $\mathbf{v}_i = (v_{i1}, v_{i2}, v_{i3})^\top$. They span a simplex $\mathbf{V} = \triangle(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$. Hence, each

point $\mathbf{p} \in \mathbb{R}^3$ can be expressed as a unique linear combination

$$\mathbf{p} = \sum_{i=0}^{3} p_i \mathbf{v}_i \quad \text{with} \quad \sum_{i=0}^{3} p_i = 1. \tag{1}$$

The quadruple $\tilde{\mathbf{p}} = (p_0, p_1, p_2, p_3)$ are the *barycentric coordinates of the point* $\mathbf{p}$ *with respect to the simplex* $\mathbf{V}$. They can be computed from

$$p_i = \frac{[\mathbf{v}_0, \ldots, \{\mathbf{v}_i, \mathbf{p}\}, \ldots, \mathbf{v}_3]}{[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]}, \quad \text{for} \quad i \in \{0, 1, 2, 3\} \tag{2}$$

with

$$[\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}] = \begin{pmatrix} 1 & 1 & 1 & 1 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{pmatrix}, \tag{3}$$

where $[\mathbf{v}_0, \ldots, \{\mathbf{v}_i, \mathbf{p}\}, \ldots, \mathbf{v}_3]$ indicates that we replace the column containing the Cartesian coordinates of the point $\mathbf{v}_i$ with those of the point $\mathbf{p}$.

Let $\mathbf{i} = (i_0, i_1, i_2, i_3) \in \mathbb{Z}_+^4$, $\tilde{\mathbf{x}} = (x_0, x_1, x_2, x_3) \in \mathbb{R}^4$ and $|\mathbf{i}| = i_0 + i_1 + i_2 + i_3$. The *Bernstein-Bézier polynomials of degree* n are

$$B_{\mathbf{i}}^n(\mathbf{x}) = \binom{n}{\mathbf{i}} \tilde{\mathbf{x}}^{\mathbf{i}} = \frac{n}{i_0! i_1! i_2! i_3!} x_0^{i_0} x_1^{i_1} x_2^{i_2} x_3^{i_3} \tag{4}$$

for all $\mathbf{i}$ such that, $|\mathbf{i}| = n$. Clearly, such a polynomial is homogeneous in the barycentric coordinates. It is well known that the $\{B_{\mathbf{i}}^n(\mathbf{x}) \colon |\mathbf{i}| = n\}$ form a basis of the linear space $\Pi_n(\mathbb{R}^3)$ of all trivariate polynomials of degree n.

Let $f \in \Pi_n(\mathbb{R}^3)$ be a trivariate polynomial of degree n. After introducing barycentric coordinates for its argument and homogenization, f can be uniquely written as

$$f(\mathbf{x}) = \sum_{|\mathbf{i}|=n} B_{\mathbf{i}}^n(\mathbf{x}) b_{\mathbf{i}}, \tag{5}$$

where the $b_{\mathbf{i}} \in \mathbb{R}$ are called the *coefficients* of the polynomial $f(\mathbf{x})$.

The derivatives of the polynomial f and their expression in the Bernstein-Bézier basis can be easily calculated using the polar form of the polynomial. This concept from multilinear algebra was introduced by L. Ramshaw to geometric modeling as the *blossoming principle* (see [11] and, more recently, [12]). See appendices A and B.

3 Normal estimation

This part is devoted to the estimation of normals for each point of a given point cloud. The normals are used later during the patch fitting (see section 5). We estimate the normal using the plane of regression for certain neighborhood of each points in the data set.

We assume that the set P of points is uniformly distributed on the surface of the solid.[3] The estimation of normals from scattered data is a standard technique for data processing (see [7, 9]). The algorithm is as follows.

Algorithm 1 (Normal estimation)

1. Find all points within a certain neighbourhood of each point $\mathbf{p}_i \in P$.
2. Estimate the direction of the normal $\mathbf{n}_i$ for $\mathbf{p}_i$ via PCA (see below).
3. Orient the generated normals $\mathbf{n}_i$ consistently, via region growing.

The three steps will now be discussed in more detail.

Step 1. The neighbourhood of each point $\mathbf{p}_i \in P$ consists of the k nearest neighbours in the data set,

$$P_i = \{\mathbf{p}_{i,0}, \ldots, \mathbf{p}_{i,k-1}\} \tag{6}$$

It can be computed using suitable algorithms from computational geometry, such as hashing and kD-trees, see [14]. We suppose that $\mathbf{p}_{i,0} = \mathbf{p}_i$ and the points in P_i are sorted by ascending distance to the point $\mathbf{p}_i$.

Step 2. The estimation of the unit direction $\mathbf{n}_i$ of the normal, relies on Principal Component Analysis (PCA) applied on the above computed neighbourhood P_i of the point. Let

$$\mathbf{a}_i = \frac{1}{k} \sum_{j=0}^{k-1} \mathbf{p}_{i,j} \tag{7}$$

be the centroid of the neighbourhood P_i. Consider the quadratic form

$$q(\mathbf{x}) = (\mathbf{x} - \mathbf{a}_i)^\top Q (\mathbf{x} - \mathbf{a}_i) \quad \text{with} \quad Q = \sum_{j=0}^{k-1} (\mathbf{p}_{i,j} - \mathbf{a}_i)(\mathbf{p}_{i,j} - \mathbf{a}_i)^\top. \tag{8}$$

The eigenvector of Q which is associated with the smallest eigenvalue is used as an estimate of the normal. It can be shown to be the normal of the plane of regression of the points P_i. The normal is normalized in order to obtain a unit vector. In the sequel, we will often refer to the estimated normal at $\mathbf{p}_i$ shortly as the normal at $\mathbf{p}_i$.

Step 3. The estimated normals may not be oriented consistently, since normals at neighbouring points may have different orientations (cf. Figure 2). We use a region–growing–type algorithm for generating a consistent orientation via systematic reorientation ("swapping") of the normals. It is based on the quantity

$$\cos \gamma_{ij} = \langle \mathbf{n}_i, \mathbf{n}_j \rangle \tag{9}$$

which is compared with a given threshold. As an example, we applied this technique to the tennis ball data, see Figure 2.

[3] More precisely, we assume, that the eigenvalues calculated during the PCA are well separated.

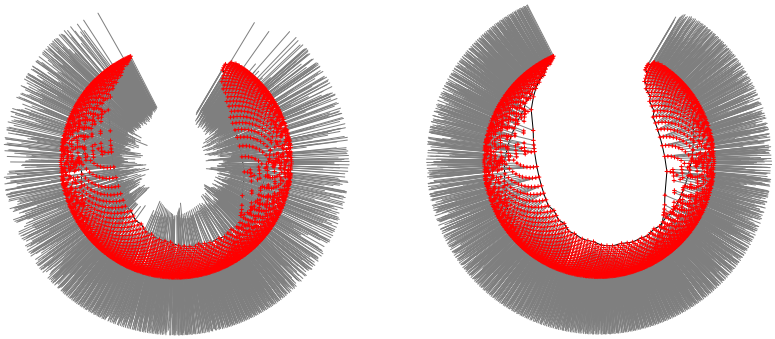

Fig. 2. Inconsistent(left) and consistent(right) orientation of normals

Clearly, the estimated tangent plane of the original surface at a point $\mathbf{p}_i$ has the equation

$$\langle \mathbf{x} - \mathbf{p}_i, \mathbf{n}_i \rangle = 0 \tag{10}$$

Implementation issues. In order to define the neighbourhood of a point $\mathbf{p}_i$, we used all points within a ball of a certain constant radius, and among those we picked the $k = 35$ closest ones. Note that the number of points in the neighbourhood might be smaller than k, especially in regions with a low sampling rate. It is recommended either to use advanced techniques of computational geometry for larger sets P to find the k nearest neighbours, or to split the input data into smaller subsets whenever possible.

Currently, our region–growing algorithm for orientation is a semi-automatic one, requiring some user interaction (e.g., adjusting the threshold). For all our examples it worked without problems.

4 Constructing boundaries

Boundary detection for triangulated point clouds is well understood. In this paper, however, we describe a meshless method, which does not assume the existence of a triangulation.[4]

The detection of boundaries in the discrete set of points is a subtle issue, which strongly depends on measurement errors in the point cloud and the distribution of the points. It consists of two major steps:

Algorithm 2 (Boundary building)

1. Detect the candidate boundary points.
2. Construct boundary polygons from boundary points.

[4] Although methods for triangulations of point clouds have recently made some progress, they are still difficult and computationally expensive. Also, they may not give the desired results, especially for complicated and/or noisy data. Therefore, meshless methods are a valuable alternative [15].

Step 1. The points on the boundary of the set P are characterized by a having a bigger distance from the centroid of their neighbourhood than the points within the inner part of the surface, see Figure 3. Consequently, the first criterion for

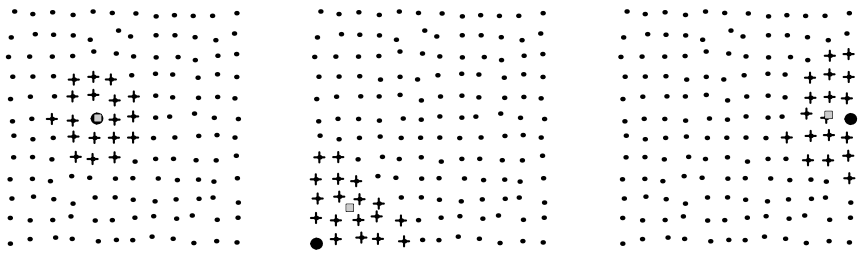

Fig. 3. Points and centroids in a point cloud for inner (left) and boundary points (middle and right). The crosses indicate the closest neighbors of the big black dot. The centroid is shown as a square.

the boundary points is

$$\|\mathbf{p}_i - \mathbf{a}_i\| \geq \varepsilon_{\text{bdist}} \tag{11}$$

for an appropriate threshold $\varepsilon_{\text{bdist}}$. Let $P_B \subseteq P$ be the set of all such points in P.

More sensitive criteria can be constructed based on the use of higher moments. This has recently been explored for feature detection [3]. Other information about the local behaviour of the point cloud can be obtained by analyzing the distribution of the eigenvalues of the matrix (8).

According to our experience, the criterion (11) works reasonably well to generate candidate points both for points on boundary and at sharp edges of the object. A finer classification can then be obtained by a local analysis in the estimated tangent plane (10), as follows.

Consider the neighbourhood P_i of a boundary point $\mathbf{p}_i$. In addition to $\mathbf{p}_i$, it contains the points $\{\mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,k-1}\}$. By projecting it into the estimated tangent plane $T_{p_i}(S)$ we get the points

$$\mathbf{R} = \{\mathbf{r}_1, \ldots, \mathbf{r}_{k-1}\}. \tag{12}$$

Let

$$w_{\mathbf{p}_i}(\mathbf{r}_l, \mathbf{r}_j) = \text{wedge}(\mathbf{r}_l \mathbf{p}_i \mathbf{r}_j) \tag{13}$$

be the wedge spanned by points $\mathbf{r}_l$ and $\mathbf{r}_j$ with the apex $\mathbf{p}_i$, and

$$\max_{l,j} \{\angle w_{\mathbf{p}_i}(\mathbf{r}_l, \mathbf{r}_j) \colon \operatorname{int} w_{\mathbf{p}_i}(\mathbf{r}_l, \mathbf{r}_j) \cap \mathbf{Q} = \emptyset\} \tag{14}$$

be the biggest angle of all those wedges which do not contain any other point from $\mathbf{R}$ (see Figure 4, left).

In the limit, if the neighbourhood became infinitesimal small, and the sampling density were arbitrarily high, the angle would be equal to π for all regular points on the boundary. For regular points in the inner part, the corresponding limit is zero.

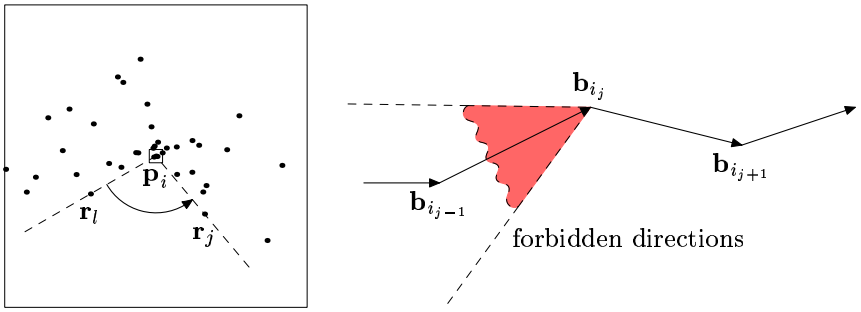

Fig. 4. Left: the maximal angle between the consecutive points. Right: boundary polygon construction. The colored wedge zone is forbidden for the next point $\mathbf{c}_{i+1}$.

In order to detect the boundary points $\mathbf{p}_i$ among the candidate points P_B, we use the following test:

- Project the neighbouring points orthogonally into the local plane of regression (the estimated tangent plane) $T_{\mathbf{p}_i}$.
- Sort them according to the polar coordinates around the point $\mathbf{p}_i$.
- Calculate the (oriented) angles of the wedges spanned by any two consecutive points with apex at $\mathbf{p}_i$ (see Figure 4), left. Let α_i be the maximum of these angles.
- Delete the point $\mathbf{p}_i$ from P_B if $\alpha_i < \alpha_0$, where α_0 is a user–defined threshold.

Clearly, this test will also delete vertices of the boundary curve with sharp inner angles. This, however, is not a problem for the hole filling application. The complexity of this algorithm is $O(Nk \log k)$.

Step 2. Now we are ready to find the boundary polygons. For the sake of simplicity, we first suppose that there are no sharp edges (no singular points and the curvature has an upper bound) along the boundary of the surface. (Such points are handled by glueing together several branches of the boundary polygon.) We use the following greedy algorithm:

Assume the set $B = \{\mathbf{b}_0, \ldots, \mathbf{b}_q\}$ contains all boundary points of set P. Points of B which are not in any constructed polygon are called *free*.

- Take a free point $\mathbf{b}_{i_0}$ and its closest free neighbour $\mathbf{b}_{i_1}$ from B.
- Extend the polygon by adding the nearest free point from B which belongs to the allowed wedges as shown in Figure 4, right. The polygon is allowed to grow in both directions.

- If there is no point to extend the polygon, or if the polygon is already closed, start another one until all the points in B are used up.

This algorithm produces the set of polygonal boundaries. If the boundary of a hole makes sharp turns, then the method will produce different polygonal segments, one for each edge. In this case, one has – as a postprocessing step – to glue the different segments together.

Clearly, the complexity of this step is $O(q^2)$. An example of the whole process is shown in the Figure 5.

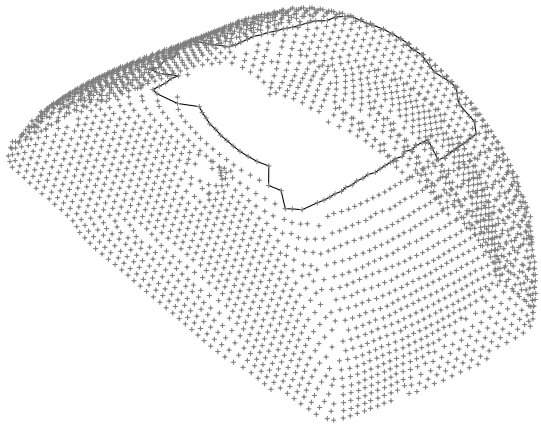

Fig. 5. Building the boundary of a hole in scanned data set

Unfortunately, there is no guarantee of the topological correctness for the boundary obtained by this algorithm. The result strongly depends on the distribution of the points and errors in P. Topologically correct solutions could be guaranteed by using so–called "snakes" (active curves) from computer vision [10]. These techniques, however, always need an initialization, which has to be provided by the user. Also, since we do not assume that the given data are triangulated, the rules for the evolution of a "snake" on a point cloud are not fully obvious. Still, this method may have some potential, and we intend to look into it in the future.

Implementation issues. Several user–defined parameters appear in the above algorithms. They strongly depends on statistical characteristics of the set P such as the average distance of closest neighbours, curvature bounds of the scanned surface, etc. They can be locally adapted according to these parameters.

The parameters are estimated according to statistical properties. The parameter $\varepsilon_{\text{bdist}}$ is set to a quarter of the average distance of the closest neighbour in the set. The parameter $\varepsilon_{\text{bangle}} \in [-1, 1]$ was chosen equal to -0.5 in our examples. It controls the size of the feasible wedge during the boundary point detection.

Note that "outliers" on the boundary of a hole may sometimes not be detected as boundary points, since its neighbourhood may not contain sufficiently many points to get reliable estimates. However, outliers have never been a problem for the further processing, and it may sometimes even be better to ignore them.

5 Algebraic surface fitting

The fitting of algebraic surfaces to given data have been discussed in several publications, see [9] and references cited therein. The method in [9] can be used to solve the following problem:

Consider a set of points $\{\mathbf{q}_0, \ldots, \mathbf{q}_r\}$ inside a tetrahedron $\triangle(\mathbf{v}_0\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3)$. Let $\mathbf{n}_i$ be a unit vector (representing the normal) associated with every point $\mathbf{q}_i$. Find an algebraic surface of degree n matching simultaneously the points and the associated normal vectors.

Considering simultaneously points and associated normals has two major advantages. First, the solution can be found by solving a linear system of equations. (Other methods require solving an eigenvalue problem instead [16].) Second, the use of the normals helps to keep unwanted branches away from the region of interest, since the resulting surface can be expected to be regular in the neighbourhood of the data. (Alternatively, regularity can always be achieved by adding suitable "tension terms" to the objective function, or by imposing monotonicity conditions or sign conditions, see [2]. While the first approach tends to "flatten" the shape, the second one requires more sophisticated solvers.)

We represent the algebraic surface as the zero set of a polynomial f of degree n in Bernstein-Bézier form over a simplex $\mathbf{V}$, see (5). The simplex $\mathbf{V}$ is chosen such that it contains the given points.

In order to fill the hole(s) in the data, we want to extend the shape of the point cloud according to the shape of the vicinity of the boundary. Thus, in order to match the boundary data by a surface

$$\mathbf{S} = \{\mathbf{x} \in \mathbf{V} \colon f(\mathbf{x}) = 0\}, \tag{15}$$

we take into account not only the points of P from the detected boundary polygon Q, but also the points in a certain tubular neighbourhood $N(Q)$. The tubular neighbourhood is approximated by a union of balls,

$$N(Q) = \bigcup_{i=0}^{r} B_{\varepsilon_{\text{tube}}}(\mathbf{q}_i), \tag{16}$$

where $B_\varepsilon(\mathbf{p})$ is the ball centered at $\mathbf{p}$ with radius ε (see Figure 6).

To each point we assign the weight μ_i for $i \in \{0, \ldots, r\}$. It is proportional to number of balls $B_{\varepsilon_{\text{tube}}}(\mathbf{q}_j)$ in (16), which contain the point $\mathbf{q}_i$. For points further away from the hole, this weight decreases. As another possibility, one may choose these weights μ_i according to the distance $d(\mathbf{p}_i, Q)$ from the boundary polygon Q. Clearly, this is more expensive to compute.

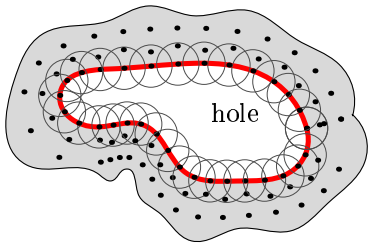

Fig. 6. Tubular neighbourhood of a boundary

The objective function has the form

$$F(\boldsymbol{b}) = w_0 D(\boldsymbol{b}) + w_1 N(\boldsymbol{b}) + w_2 T_1(\boldsymbol{b}) \ \{ + w_3 T_2(\boldsymbol{b}) \} \tag{17}$$

where

$$D(\boldsymbol{b}) = \sum_{k=0}^{r} \mu_k f(\mathbf{q}_k)^2, \tag{18}$$

$$N(\boldsymbol{b}) = \sum_{k=0}^{r} \mu_k \|\nabla f(\mathbf{q}_k) - \mathbf{n}_k\|^2, \tag{19}$$

$$T_1(\boldsymbol{b}) = \int_{\mathbf{V}} f_{xx}^2 + f_{yy}^2 + f_{zz}^2 + 2f_{xy}^2 + 2f_{xz}^2 + 2f_{yz}^2 \mathrm{d}V, \tag{20}$$

$$T_2(\boldsymbol{b}) = \int_{\mathbf{V}} f_{xxx}^2 + \cdots + f_{zzz}^2 \mathrm{d}V \tag{21}$$

and $\boldsymbol{b} = (b_{\mathbf{i}})_{|\mathbf{i}|=n}$ are the coefficients in (5) and $w_0, w_1, w_2, w_3 \in \mathbb{R}_+$ are constant weights. The term $T_1(\boldsymbol{b})$ ($T_2(\boldsymbol{b})$) is a "tension term", which can be used to pull the solution towards a plane (quadric surface). It should be used only if it is needed for avoiding singularities (unwanted branches of the algebraic surface). Otherwise, one may set $w_2 = 0$ ($w_3 = 0$). Clearly, the value of the tension term also depends on the shape of the simplex $\mathbf{V}$.

Since the objective function is a quadratic positive definite function of the coefficients $\boldsymbol{b}$, we can find its minimum by solving the linear system

$$\nabla F(\boldsymbol{b}) = \mathbf{0}. \tag{22}$$

The resulting formulas have been gathered in Appendix C.

As an example, we applied the fitting procedure to the upper part of the babyphone point cloud (see Figure 1, left), in order to fill it with an algebraic surface of degree 4. The choice of the degree is a useful compromise between flexibility and number of degrees of freedom (shape parameters). According to our experience, degree 4 was sufficient in most cases. The part of the surface corresponding to the inner part of the hole is shown in Figure 7. It was produced with the weights $w_0 = 1000$, $w_1 = 550.0$ and $w_2 = 1.0$.

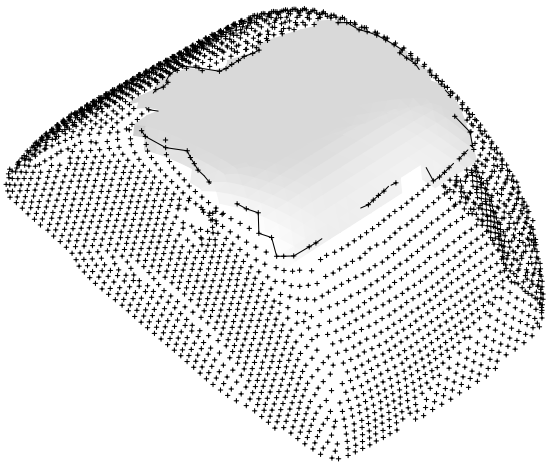

Fig. 7. Fitting a surface to the boundary data of the upper hole in the babyphone data set.

Remark 1. If the weight $w_2 \neq 0$, then the algebraic surface fitting *reproduces planes*. If $w_2 = 0$, but $w_3 \neq 0$, then it *reproduces spheres and circular cylinders*. Indeed, if both the points and the associated normals are sampled from an algebraic surface, whose gradients have the same length everywhere, then this algebraic surface is the unique minimizer of the objective function. Clearly, planes, circular cylinders and spheres enjoy this property. Other quadrics (such as cones, ellipsoids, etc.) are therefore generally not preserved.

Remark 2. If $n \geq 4$ and $w_2 = w_3 = 0$, the data the data taken from a sphere (or circular cylinder) produces a singular system (22), since any product of the equation of the sphere (or cylinder) with a cocentric sphere (or coaxial cylinder) would be a solution.

For holes with more complicated geometry, it is necessary to use algebraic spline surfaces (instead of single patches), and/or to split the data.

6 Generating the points

In order to finish the filling of the hole, we need to generate sample points from the surface, which has been fitted to the boundary data. Clearly, we need only points from the "inner part" of the hole. More precisely, let $\mathbf{S} \subset \mathbb{R}^3$ be an algebraic surface (or a part of it) defined by the equation $f(\mathbf{x}) = 0$, with a chosen orientation. Let $\mathbf{c}\colon [0,1] \mapsto \mathbf{S}$ be a positively oriented simple closed curve on the surface $\mathbf{S}$. We have to find a finite subset of points $P' \subset \mathbf{S}$ such that all points are inside the region of $\mathbf{S}$ enclosed by the curve $\mathbf{c}$.

Our algorithm is based on a triangulation of the surface $\mathbf{S}$, and on approximate geodesic offsets of the boundary curve $\mathbf{c}$.

Algorithm 3 (Generating the points)

1. Approximate the given surface $\mathbf{S}$ in region of interest by a triangulation $\mathcal{T}$.
2. Project the curve $\mathbf{c}$ onto the triangulation and find the region $\mathcal{R} \subset \mathcal{T}$ enclosed by the projected curve.
3. Generate a set $P_{\mathcal{R}}$ of points in the $\mathcal{R}$ and compute the set $P_{\mathbf{S}}$ of their corresponding footpoints on $\mathbf{S}$.
4. Choose a uniform subset of $P_{\mathbf{S}}$ by geodesic offsetting, in order to get P'.

Step 1. We have used the algorithm of "marching triangles". The global strategy can be found in [1]. In general, this algorithm provides relatively nice triangulations; no post-processing is needed.

The original algorithm had to be modified in order to avoid thin triangles during the approximation as far as possible. Similarly, the final phase of the algorithm ("connecting the cracks") has been improved.

Step 2. The polygon $\mathbf{c}_0\mathbf{c}_1 \dots \mathbf{c}_r$ which approximates the boundary curve $\mathbf{c}$ is projected onto the triangulation $\mathcal{T}$. We detect all triangles of $\mathcal{T}$ which are enclosed by the projected boundary curve; they form the domain $\mathcal{R}$.

Step 3. The generation of the points on the $\mathcal{R}$ and the their projection back to the $\mathbf{S}$ is straightforward. Since the triangles have approximately the same size, we generate – for each one – the same number of points.

Step 4. We approximate the geodesic offsets of the boundary curve. (For more detail on numerical methods of computing geometric offsets, see e.g. [13].) It is assumed that we have a "sufficiently dense" set of points on $\mathbf{S}$. In addition, the user has to specify a radius ε_r for offsetting; it should be equal to the average point distance in the point cloud.

- Delete from $P_{\mathbf{S}}$ all the points in a tubular neighbourhood of the boundary $\mathbf{c}$.
- Repeat until $P_{\mathbf{S}}$ is empty:
  For every point on the boundary find the closest point in $P_{\mathbf{S}}$, and add it to the set of boundary points, see Figure 8. Delete all points within a tubular neighbourhood of the new set from $P_{\mathbf{S}}$.

The method has been applied to the two data sets from Figure 1, see Figures 9, 10. As one can see from the tennis ball example, the method for algebraic surface fitting is able to reproduce spheres. We used the degree $n = 4$ and weights $w_0 = 10,000$, $w_1 = 1.0$ and $w_2 = 0.0001$.[5]

[5] Due to $w_2 \ll w_1 \ll w_0$, the influence of $T_2(\boldsymbol{b})$ is relatively small. Consequently, the method almost exactly reproduces sphere, even if the condition of Remark 1 is violated.

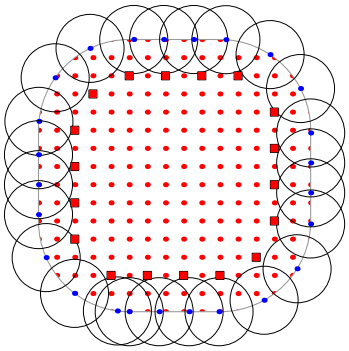

Fig. 8. Generating of points approximating geodesic offsets.

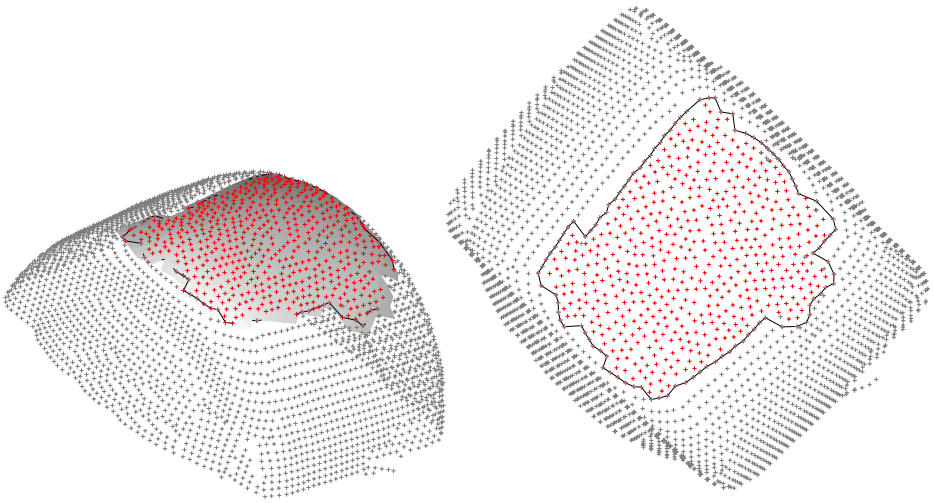

Fig. 9. Filling the hole in the babyphone data.

Implementation issues. All the tubular neighbourhoods have the radius ε_r. They are always approximated by the union of balls around the data.

The size of the triangle should be smaller than the offsetting radius ε_r. If this is satisfied, then the first part of Step 4 excludes the points which are on the wrong side of the boundary.

Remark 3. The number of generated points is proportional to the number of triangles. Hence, it is appropriate to find a balance between the number of points generated per triangle and the size of generic triangle chosen for the approximation of the surface. For medium-sized holes, the number of generated points is reasonable to deal with. In the case of bigger holes, it is recommended to use efficient methods of computational geometry (such as hashing) to delete unwanted points from the generated set.

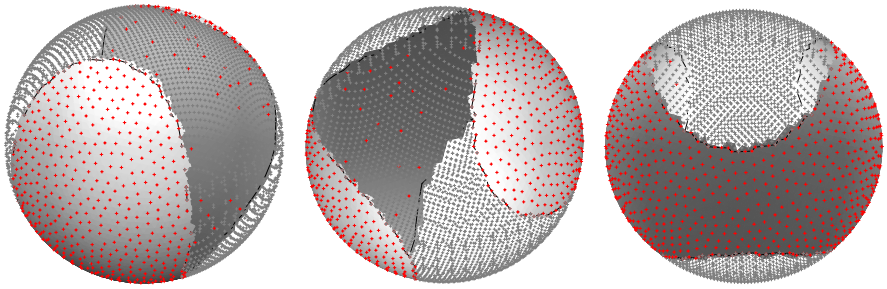

Fig. 10. Reconstructing the tennis ball.

7 Conclusions

We described a method for filling holes in point clouds. After detecting the holes and their boundaries, we fitted an algebraic surface patch to the neighbourhood of the boundary. Finally, auxiliary points filling the hole were generated by approximating the geodesic offsets of the boundary.

The use of an algebraic fit has several advantages. First, the surface can be obtained without assuming the existence of a suitable parameterization of the data. Second, the method can be applied to holes of more general shapes (such as the tennis ball data, see Figure 10), and more general topologies. For instance, holes with a cylindrical shape can be handled by our algorithm. Finally, the method is able to reproduce planes, circular cylinders and spheres, which are clearly important for engineering applications.

Currently, we implemented only a single patch of an algebraic surface. In order to fit more complicated shapes, that cannot be covered by a single algebraic patch, the use of algebraic splines (such as defined by Clough-Tocher, Powell-Sabin or Bajaj macroelements [2, 8]) will be more appropriate. In this case, however, the use of suitable tension terms will become more important, since no geometric information is present in the inner part of the hole.

A Blossoming

We recall some results associated with the blossoming principle (polar forms). There is a unique bijective mapping between polynomials of degree n and multilinear, symmetric functions. Let f be a polynomial of degree n. Then, there is a multilinear symmetric function

$$f_*\colon \underbrace{\mathbb{R}^4 \times \cdots \times \mathbb{R}^4}_{n\text{-times}} \mapsto \mathbb{R} \tag{23}$$

such that

$$D_{\boldsymbol{\eta}_1,\ldots,\boldsymbol{\eta}_q} f(\mathbf{u}) = \frac{n!}{(n-q)!} f_*(\tilde{\boldsymbol{\eta}}_1 \ldots \tilde{\boldsymbol{\eta}}_q \hat{\mathbf{u}}^{n-q}) \tag{24}$$

where $\hat{\mathbf{u}} = (1, u_1, \ldots, u_d)^\top$ and $\tilde{\boldsymbol{\eta}}_i = (0, \eta_{i1}, \ldots, \eta_{id})^\top$ are the projective extensions of points and vectors. Note that the derivative of order zero is the value of the polynomial itself; the formula is valid in this case, too. Hence, the polar form can be used to evaluate all directional derivatives of polynomial f. Clearly, the existence of a polynomial from a given multilinear symmetric function follows directly.

A multilinear function (23) is determined by $\binom{n+3}{3}$ coefficients $b_{\mathbf{i}}$ which can be seen as the values of f_* on the $e_0^{i_0} e_1^{i_1} e_2^{i_2} e_3^{i_3}$ for an arbitrary but fixed basis $\{e_0, \ldots, e_3\} \in \mathbb{R}^4$.

B Multiplying multivariate polynomials in Bernstein-Bézier form

In the sequel we will need multiplication formulas for polynomials in Bernstein-Bézier form. We briefly recall the algorithm, see [4] for a more general approach (also covering the composition of polynomials in Bernstein–Bézier form). Let

$$f(\mathbf{x}) = \sum_{|\mathbf{i}|=n} B_{\mathbf{i}}^n(\mathbf{x}) b_{\mathbf{i}} \quad \text{and} \quad g(\mathbf{x}) = \sum_{|\mathbf{j}|=m} B_{\mathbf{j}}^m(\mathbf{x}) c_{\mathbf{j}} \tag{25}$$

be two polynomials of degree n and m. Their product in the corresponding Bernstein-Bézier basis,

$$h(\mathbf{x}) = \sum_{|\mathbf{k}|=m+n} B_{\mathbf{k}}^{n+m}(\mathbf{x}) d_{\mathbf{k}}, \tag{26}$$

has the coefficients

$$d_{\mathbf{k}} = \sum_{\mathbf{i}+\mathbf{j}=\mathbf{k}, |\mathbf{i}|=n, |\mathbf{j}|=m} \frac{\binom{|\mathbf{i}|}{\mathbf{i}} \binom{|\mathbf{j}|}{\mathbf{j}}}{\binom{|\mathbf{k}|}{\mathbf{k}}} b_{\mathbf{i}} c_{\mathbf{j}}. \tag{27}$$

Finally, we recall the identity

$$\int_{\mathbf{V}} f(\mathbf{x}) \mathrm{d}V = \frac{\mathrm{vol}(\mathbf{V})}{\binom{n+3}{3}} \sum_{|\mathbf{i}|=n} b_{\mathbf{i}}, \quad \text{where} \quad \mathrm{vol}(\mathbf{V}) = \frac{1}{3!}[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3], \tag{28}$$

which is needed for evaluating the tension terms.

C Minimizing the objective function

The objective function (17) is minimized by solving (22). More precisely, this leads to

$$\frac{\partial}{\partial b_{\mathbf{i}}} F(\boldsymbol{b}) = w_0 A_0 + w_1 A_1 + w_2 A_2, \tag{29}$$

where

$$A_0 = \sum_{k=0}^{r} 2\mu_k f(\mathbf{q}_k) B_{\mathbf{i}}^n(\mathbf{q}_k), \tag{30}$$

$$A_1 = \sum_{k=0}^{r} 2\mu_i \langle \nabla f(\mathbf{q}_k) - \mathbf{n}_k, \frac{\partial}{\partial b_{\mathbf{i}}} \nabla f(\mathbf{q}_k) \rangle \tag{31}$$

and

$$\begin{aligned} A_2 = \int_{\mathbf{V}} \Bigg(& 2f_{xx}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{xx}(\mathbf{x}) + 2f_{yy}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{yy}(\mathbf{x}) + 2f_{zz}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{zz}(\mathbf{x}) + \\ & 4f_{xy}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{xy}(\mathbf{x}) + 4f_{xz}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{xz}(\mathbf{x}) + 4f_{yz}(\mathbf{x}) \frac{\partial}{\partial b_{\mathbf{i}}} f_{yz}(\mathbf{x}) \Bigg) \mathrm{d}V, \end{aligned} \tag{32}$$

Equation (30) can be rewritten as

$$A_0 = \sum_{|\mathbf{j}|=n} \left(\sum_{k=0}^{r} 2\mu_k B_{\mathbf{j}}^n(\mathbf{q}_k) B_{\mathbf{i}}^n(\mathbf{q}_k) \right) b_{\mathbf{j}}. \tag{33}$$

In order to simplify (31), we write – according to (24) –

$$\begin{aligned} D_{\boldsymbol{\eta}} f(\mathbf{x}) &= n f_*(\tilde{\boldsymbol{\eta}} \hat{\mathbf{x}}^{n-1}) \\ &= \sum_{|\mathbf{j}|=n} \underbrace{n(\eta_0 B_{\mathbf{j}-\mathbf{e}_0}^{n-1}(\mathbf{x}) + \eta_1 B_{\mathbf{j}-\mathbf{e}_1}^{n-1}(\mathbf{x}) + \eta_2 B_{\mathbf{j}-\mathbf{e}_2}^{n-1}(\mathbf{x}) + \eta_3 B_{\mathbf{j}-\mathbf{e}_3}^{n-1}(\mathbf{x}))}_{D_{\boldsymbol{\eta}} B_{\mathbf{j}}^n(\mathbf{x})} b_{\mathbf{j}}, \end{aligned} \tag{34}$$

where $\boldsymbol{\eta}$ is the representation of any vector in barycentric coordinates with respect to the tetrahedron $\mathbf{V}$ and polynomials with negative indices vanish identically. Consequently,

$$A_1 = 2n \sum_{k=0}^{r} \mu_k \Big(n \sum_{|\mathbf{j}|=n} \langle \nabla B_{\mathbf{j}}^n(\mathbf{q}_k), \nabla B_{\mathbf{i}}^n(\mathbf{q}_k) \rangle b_{\mathbf{j}} - \langle \mathbf{n}_{\mathbf{j}}, \nabla B_{\mathbf{i}}^n(\mathbf{q}_k) \rangle \Big) \tag{35}$$

with

$$\nabla B_{\mathbf{i}}^n(\mathbf{q}_k) = (D_{\mathbf{x}} B_{\mathbf{i}}^n(\mathbf{q}_k), D_{\mathbf{y}} B_{\mathbf{i}}^n(\mathbf{q}_k), D_{\mathbf{z}} B_{\mathbf{i}}^n(\mathbf{q}_k)). \tag{36}$$

Similarly as in (34) one gets

$$D_{\boldsymbol{\eta}_1 \boldsymbol{\eta}_2} f(\mathbf{x}) = \sum_{|\mathbf{j}|=n} D_{\boldsymbol{\eta}_1 \boldsymbol{\eta}_2} B_{\mathbf{j}}^n(\mathbf{x}) b_{\mathbf{j}}$$

where by (24)

$$D_{\boldsymbol{\eta}_1 \boldsymbol{\eta}_2} B_{\mathbf{j}}^n(\mathbf{x}) = n(n-1) f_{\mathbf{j}*}(\tilde{\boldsymbol{\eta}}_1 \tilde{\boldsymbol{\eta}}_2 \hat{\mathbf{x}}^{n-2}).$$

Further, if we denote

$$\Delta_{\mathrm{diag}} B_{\mathbf{i}}^n(\mathbf{x}) = (D_{\mathbf{xx}} B_{\mathbf{i}}^n(\mathbf{x}), D_{\mathbf{yy}} B_{\mathbf{i}}^n(\mathbf{x}), D_{\mathbf{zz}} B_{\mathbf{i}}^n(\mathbf{x}))^\top \tag{37}$$

and

$$\Delta_{\text{offdiag}} B_{\mathbf{i}}^n(\mathbf{x}) = (D_{\mathbf{xy}} B_{\mathbf{i}}^n(\mathbf{x}), D_{\mathbf{xz}} B_{\mathbf{i}}^n(\mathbf{x}), D_{\mathbf{yz}} B_{\mathbf{i}}^n(\mathbf{x}))^\top, \tag{38}$$

the term (32) can be rewritten as

$$\begin{aligned} A_2 = 2 \int_{\mathbf{V}} \sum_{|\mathbf{j}|=n} \big(& \langle \Delta_{\text{diag}} B_{\mathbf{j}}^n(\mathbf{x}), \Delta_{\text{diag}} B_{\mathbf{i}}^n(\mathbf{x}) \rangle \\ & + 2 \langle \Delta_{\text{offdiag}} B_{\mathbf{j}}^n(\mathbf{x}), \Delta_{\text{offdiag}} B_{\mathbf{i}}^n(\mathbf{x}) \rangle \big) b_{\mathbf{j}} \, \mathrm{d}V \end{aligned} \tag{39}$$

and evaluated using (25)–(28).

Acknowledgements

This research was supported by the Austrian Science Fund (FWF) through the SFB F013 "Numerical and Symbolic Scientific Computing" at Linz, project 15. The authors wish to thank the referees for their comments which have helped to improve the paper.

References

1. S. Akkouche and E. Galin. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum*, 20(2):67–80, 2001.
2. C. L. Bajaj. Implicit surface patches. In J. Bloomenthal, editor, *Introduction to implicit surfaces*. Morgan Kaufmann, San Francisco, 1997.
3. U. Clarenz, M. Rumpf, and A. Telea. Robust feature detection and local classification for surfaces based on moment analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2003. submitted, available online at `http://numerik.math.uni-duisburg.de/research/publications.htm`.
4. T. DeRose, R.N. Goldman, H. Hagen, and S. Mann. Functional composition algorithms via blossoming. *ACM Trans. Graph.*, 12(2):113–135, 1993.
5. B. Curless et al. The Digital Michelangelo Project. `http://graphics.stanford.edu/projects/mich/`, 2000.
6. G. Farin, J. Hoschek, and M.-S. Kim, editors. *Handbook of computer aided geometric design*. North-Holland, Amsterdam, 2002.
7. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.
8. J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. AK Peters, 1993.
9. B. Jüttler and A. Felis. Least-squares fitting of algebraic spline surfaces. *Adv. Comput. Math.*, 17(1-2):135–152, 2002.
10. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.
11. L. Ramshaw. Blossoms are polar forms. *Comput. Aided Geom. Des.*, 6(4):323–358, 1989.
12. L. Ramshaw. On multiplying points: The paired algebras of forms and sites. SRC Research Report #169, COMPAQ Corp., 2001.

13. T. Rausch, F.-E. Wolter, and O. Sniehotta. Computation of medial curves on surfaces. In T. Goodman and R. Martin, editors, *The mathematics of surfaces VII*, pages 43–68. Information Geometers, Ltd., 1997.
14. J.-R. Sack and J. Urrutia, editors. *Handbook of computational geometry.* North-Holland, Amsterdam, 2000.
15. R. Schaback. Remarks on meshless local construction of surfaces. In R. Cipolla and R. Martin, editors, *The mathematics of surfaces IX. Proceedings of the 9th IMA conference Cambridge.*, pages 34–58. Springer, London, 2000.
16. R. Taubin. Estimation of planar curves, surfaces, and non-planar space curves defined by implicit equations with applications to edge and range image segmentation. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 13:1115–1139, 1991.
17. K. Tucholsky. Zur soziologischen Psychologie der Löcher. In *Zwischen Gestern und Morgen.* Rohwolt, Hamburg, 1952.
18. T. Varady, R.R. Martin, and J. Cox. Reverse engineering of geometric models - an introduction. *Comput.–Aided Des.*, 29(4):255–268, 1997.