ELSEVIER

Computer-Aided Design 33 (2001) 593–603

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Parametrization for surface fitting in reverse engineering

L.A. Piegl[a,*], W. Tiller[b]

[a]Department of Computer Science and Engineering, University of South Florida, 4202 Fowler Avenue, ENG 118, Tampa, FL 33620, USA
[b]GeomWare, Inc., 3035 Great Oak Circle, Tyler, TX 75703, USA

Received 29 May 2000; revised 31 July 2000; accepted 1 August 2000

Abstract

Given four boundary curves and a set of random points lying on a surface patch, a method for assigning parameters to these points is presented. The algorithm uses various base surfaces to project the points onto these surfaces to recover the parameters based on the surfaces' underlying parametrization. Several techniques for speeding up the time consuming projection process are also presented. A thinning method is introduced as well to select a subset of the points that may have been sampled at a much higher rate than necessary. The thinning is based on the geometry of the base surface and relies on a meaningful geometric tolerance. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Surface fitting; Reverse engineering; NURBS

1. Introduction

Object reconstruction from sampled data forms an important part of obtaining the design parameters of an existing object. Graphical models [1–3,7–14] as well as engineering parts [15,27,28,30–33] must be faithfully reconstructed, that is, some form of representation is sought that defines an object that does not deviate from the points more than a given error measure. The representation can be discrete, e.g. a set of triangles, or continuous, e.g. a B-rep model. The focus of this paper is to obtain a B-rep model through surface fitting to subsets of points. Fitting surfaces to randomly distributed data has been the subject of significant interest [4–6,10,16–25,29,33]. Using B-splines, the critical issue of such fitting process is the computation of parameters and the knot vectors. This paper deals with the first issue. More precisely, the following problem is dealt with. Assume that the data has been split up into rectangular regions bounded by curves. That is, given four boundary NURBS curves (Fig. 1)

$$\mathbf{C}_k(u) = \sum_i N_{i,p}(u)\mathbf{P}_{k,i} \qquad k = 0, 1$$

$$\mathbf{C}_\ell(v) = \sum_j N_{j,q}(v)\mathbf{P}_{\ell,j} \qquad \ell = 0, 1$$

a set of randomly distributed points (it is assumed that the points on the boundaries are stripped off)

$$\mathbf{Q}_i \qquad i = 0, \ldots, N$$

(in Fig. 1 $N = 11,516$) a set of parameters

$$(u_i, v_i) \qquad i = 0, \ldots, N$$

are sought that best represent the points' position within the rectangular domain, i.e. the data points are assumed in the NURBS representation at these parameters. A viable solution to this problem is to start with a base surface, fitted somehow to the boundary curves and/or (some of) the points, and then projecting the data points to this surface to obtain the parameters of the projected points. Theoretically this all sounds easy to do, however, there are many technical hurdles that can render this base surface based approach completely unusable.

The issue of parametrization for curve and surface fitting has been a difficult one since the advent of computers in design engineering. There is no known best parametrization, and even after many decades of research some data sets just cannot be dealt with adequately. This paper does not claim to solve the parametrization problem either. Especially not for random surface points. The aim is to give the designer a set of tools by which he can obtain good parameters at a reasonable cost. It is our opinion that the parametrization process cannot be completely automatic. The user may have to choose different options for different data sets, and he may even have to iterate a few times to obtain better results.

The contribution of this paper is twofold: (1) the

* Corresponding author. Tel.: +1-813-974-5234; fax: +1-813-974-5456.
E-mail addresses: piegl@aol.com (L.A. Piegl), geomware@gower.net (W. Tiller).

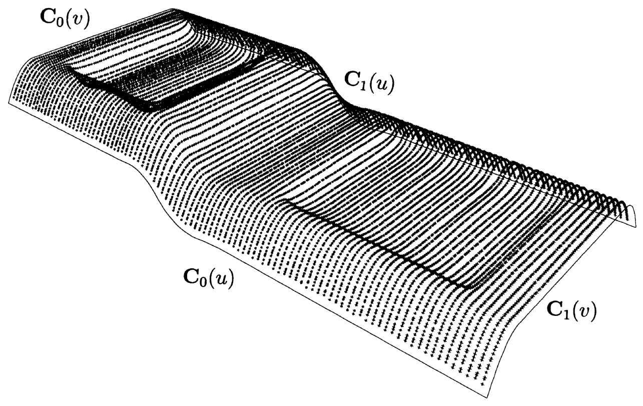


Fig. 1. Boundary curves and data points.

introduction of various base surfaces to capture more detail inherent in the data; and (2) a methodology to gain significant speed-ups during point projection. The paper is organized as follows. Section 2 deals with base surfaces, and Section 3 is devoted to obtaining stability and speed-ups during point projection. A Section 4 closes the paper.

2. Base surfaces

The general idea of obtaining parameters is to use a surface attached to the input parameters [21]. In the sections that follow, we investigate three kinds of surfaces: (1) the bilinearly blended Coons patch; (2) the bicubic Coons patch; and (3) the tensor-product surface.

2.1. Bilinear Coons

The simplest base surface is the so called bilinearly blended Coons surface. Using NURBS, the surface is expressed as follows [26]:

$$\mathbf{S}(u,v) = \mathbf{L}_u(u,v) + \mathbf{L}_v(u,v) - \mathbf{T}(u,v)$$

where

- $\mathbf{L}_u(u,v)$ is a linearly lofted surface (also known as a ruled surface) in the u-direction;
- $\mathbf{L}_v(u,v)$ is a linearly lofted surface in the v-direction; and
- $\mathbf{T}(u,v)$ is a tensor product surface, i.e. it is a bilinear surface defined by the four corner points.

Once the two ruled surfaces and the bilinear surface are constructed, they are made compatible, and the control points of $\mathbf{S}(u,v)$ are obtained by adding the control points of the ruled surfaces and subtracting those of the bilinear patch.

The advantages of using a bilinear Coons patch are:

- it is very easy to compute;
- the computation is stable and fast; and
- the surface is well parametrized.

If the data set is simple, and in many cases it is (a good reverse engineering package includes a segmentation module that subdivides the data set into very simple patches), then the bilinear Coons patch is the best base surface we can think of. However, one disadvantage with this patch is that it tends to miss internal detail, as illustrated in Fig. 2. If the segmentation process failed to do a good job, then the bilinear Coons may provide inadequate parameters.

2.2. Bicubic Coons

The idea behind using a bicubic patch is to include some internal detail into the base surface. The bicubic Coons patch has the same form as the bilinear except that the lofted surfaces are not linearly but cubically blended. To do so, we need cross derivatives. The computation of such a surface is quite involved, therefore only the main ideas are outlined below.

Assume now that in addition to the boundary curves some number of internal points are also selected. Fixing the number of control points and the degrees in both directions, the unknown entities are the cross derivatives

$$\mathbf{D}_k(u) = \sum_{i=0}^{n} M_{i,p}(u)\mathbf{P}^v_{k,i} \qquad k = 0,1$$

$$\mathbf{D}_\ell(v) = \sum_{j=0}^{m} M_{j,q}(v)\mathbf{P}^u_{\ell,j} \qquad \ell = 0,1$$

where, for example, $\mathbf{D}_1(u)$ is a cross derivative field across the boundary $\mathbf{C}_1(u)$, i.e. it is a function of u and its vectors, evaluated at various parameters, are the derivatives pointing across the boundary curve. To compute these cross derivatives, the unknown control points

$$\mathbf{P}^v_{k,i} \;\; k = 0,1 \quad i = 0,\ldots,n$$

$$\mathbf{P}^u_{\ell,j} \;\; \ell = 0,1 \quad j = 0,\ldots,m$$

are sought. For simplicity, the surface $\mathbf{S}(u,v)$, at fixed

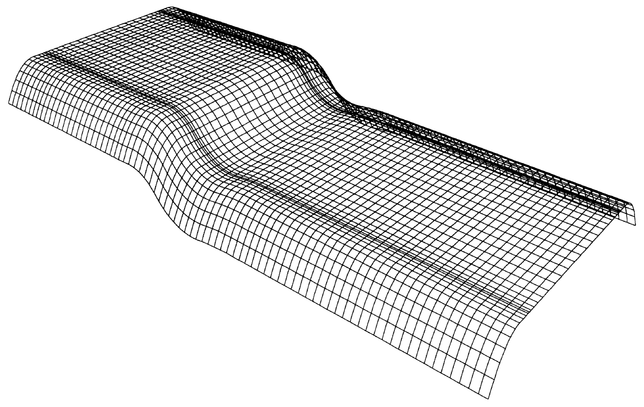


Fig. 2. Bilinearly blended Coons base surface.

parameters, is expressed using cubic Hermite polynomials

$$H_1(u) = 1 - 3u^2 + 2u^3$$

$$H_2(u) = 3u^2 - 2u^3$$

$$H_3(u) = u - 2u^2 + u^3$$

$$H_4(u) = -u^2 + u^3$$

defined over the domain [0,1]. If the surface is defined over a general domain of $[u_s, u_e] \times [v_s, v_e]$, then the parameter transformations

$$s = \frac{(u - u_s)}{\mathrm{d}u} \qquad t = \frac{(v - v_s)}{\mathrm{d}v} \qquad \mathrm{d}u = u_e - u_s$$

$$\mathrm{d}v = v_e - v_s$$

bring the domains to the unit span. Note that

$$H_1(u) = H_1(s) \qquad H_2(u) = H_2(s) \qquad H_3(u) = \mathrm{d}u H_3(s)$$

$$H_4(u) = \mathrm{d}u H_4(s)$$

Now, for a fixed parameter pair (u, v), the bicubic surface is written as

$$\mathbf{S}(u, v) = \mathbf{L}_u(u, v) + \mathbf{L}_v(u, v) - \mathbf{T}(u, v)$$

$$\mathbf{L}_u(u, v) = H_1(s)\mathbf{C}_1(v) + H_2(s)\mathbf{C}_2(v) + \mathrm{d}u H_3(s)\mathbf{D}_1(v) + \mathrm{d}u H_4(s)\mathbf{D}_2(v)$$

$$\mathbf{L}_v(u, v) = H_1(t)\mathbf{C}_1(u) + H_2(t)\mathbf{C}_2(u) + \mathrm{d}v H_3(t)\mathbf{D}_1(u) + \mathrm{d}v H_4(t)\mathbf{D}_2(u)$$

$$\mathbf{T}(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} B_{i,3}(u) B_{j,3}(v) \mathbf{P}_{i,j}$$

where $B_{i,3}(u)$ and $B_{j,3}(v)$ are the cubic Bernstein polynomials for the bicubic tensor product patch $\mathbf{T}(u, v)$. Since the boundary curves are known, the control points of $\mathbf{T}(u, v)$ can be expressed in terms of the surface derivatives at the corners. For example

$$\mathbf{P}_{00} = \mathbf{S}_{00}$$

$$\mathbf{P}_{10} = \mathbf{S}_{00} + \frac{\mathrm{d}u}{3}\mathbf{S}_{00}^{u}$$

$$\mathbf{P}_{11} = \frac{\mathrm{d}u\,\mathrm{d}v}{9}\mathbf{S}_{00}^{uv} + \mathbf{P}_{01} + \mathbf{P}_{10} - \mathbf{P}_{00}$$

where $\mathbf{S}_{00}^{u}$ and $\mathbf{S}_{00}^{uv}$ denote partial derivatives with respect to u, and u and v, respectively. Using the cross derivatives, the mixed partials are also expressed as

$$\mathbf{S}_{00}^{uv} = \frac{q}{\mathrm{d}v_s}(\mathbf{P}_{0,1}^{u} - \mathbf{P}_{0,0}^{u}) = \frac{p}{\mathrm{d}u_s}(\mathbf{P}_{0,1}^{v} - \mathbf{P}_{0,0}^{v})$$

where $\mathrm{d}u_s$ and $\mathrm{d}v_s$ denote the first nonzero knot spans of the cross derivatives in u- and v-directions, respectively. Taking $\mathbf{S}_{00}^{uv}$ to be the average of the two expressions, and substituting the equations of $\mathbf{P}_{00}, \mathbf{P}_{10}$ and $\mathbf{S}_{00}^{uv}$ into the equation of $\mathbf{P}_{11}$ one gets

$$\mathbf{P}_{11} = \frac{\mathrm{d}u\,\mathrm{d}v}{18}\left[\frac{p}{\mathrm{d}u_s}(\mathbf{P}_{0,1}^{v} - \mathbf{S}_{00}^{v}) + \frac{q}{\mathrm{d}v_s}(\mathbf{P}_{0,1}^{u} - \mathbf{S}_{00}^{u})\right] + \frac{\mathrm{d}v}{3}\mathbf{S}_{00}^{v} + \frac{\mathrm{d}u}{3}\mathbf{S}_{00}^{u} + \mathbf{S}_{00}$$

Similar results hold for the remaining inner control points $\mathbf{P}_{12}, \mathbf{P}_{21}$ and $\mathbf{P}_{22}$. Now, substituting $\mathbf{P}_{i,j}$ into the Coons formulation above, the only unknown elements are the control points of the cross derivatives. Assuming that the subset of data points

$$\mathbf{Q}_j \qquad j \in \mathscr{INDEXSET}$$

had parameters assigned, ($\mathscr{INDEXSET}$ denotes a random selection of indexes of the set $\{0, \ldots, N\}$), an overdetermined system of equations

$$\mathbf{Q}_j = \mathbf{S}(u_j, v_j) \qquad j \in \mathscr{INDEXSET}$$

can be set up and solved for the unknown control points. The system represents an unconstrained set of equations. In order to compute the Coons patch properly, twist compatibility must be ensured at the corners. This gives rise to the following constraint equations:

$$\mathbf{P}_{0,1}^{u} = \mathbf{S}_{00}^{u} + \frac{p\,\mathrm{d}v_s}{q\,\mathrm{d}u_s}(\mathbf{P}_{0,1}^{v} - \mathbf{S}_{00}^{v})$$

$$\mathbf{P}_{0,m-1}^{u} = \mathbf{S}_{01}^{u} - \frac{p\,\mathrm{d}v_e}{q\,\mathrm{d}u_s}(\mathbf{P}_{1,1}^{v} - \mathbf{S}_{01}^{v})$$

$$\mathbf{P}_{1,1}^{u} = \mathbf{S}_{10}^{u} + \frac{p\,\mathrm{d}v_s}{q\,\mathrm{d}u_e}(\mathbf{S}_{10}^{v} - \mathbf{P}_{0,n-1}^{v})$$

$$\mathbf{P}_{1,m-1}^{u} = \mathbf{S}_{11}^{u} - \frac{p\,\mathrm{d}v_e}{q\,\mathrm{d}u_e}(\mathbf{S}_{11}^{v} - \mathbf{P}_{1,n-1}^{v})$$

where $\mathrm{d}u_e$ and $\mathrm{d}v_e$ denote the last nonzero knot spans of the cross derivatives in u- and v-directions, respectively. Putting the constrained and the above unconstrained systems together, the system can be solved using Lagrange multipliers as detailed in [26].

In order to make the above scheme useful for base surface computation, the following issues must be addressed:

- *How many points to select?* Practical experience shows that the number of points needed for a base surface that follows the data reasonably faithfully, heavily depends on the data. Simple data sets need no more than 10% of the original points, whereas more complicated ones may require up to 70%.
- *How to select the points?* Since the method computes cross-derivatives, it may appear that points close to the

boundaries are better than those in the interior. Again, practical experience shows that this is not always the case. In our implementation we chose randomly distributed points.
- *How many control points are necessary?* Again, this is data dependent. At first, it may be a good idea to select the numbers of control points from the boundary curves. If the surface is not adequate, this number can be increased (sometimes decreased).
- *Where do the parameters come from?* The easiest way to deal with this is to put a bilinear Coons patch to the boundary curves, select a subset of points, and compute the parameters by projecting them onto the bilinear Coons patch.
- *How to choose the knots?* The best way to select knots is to use the ones of the boundary curves. This will ensure that no additional data is generated when computing cross derivatives.

The above scheme works very well on most of the data sets exhibiting the kind of simplicity one expects from good segmentation algorithms. Unfortunately, some surface patches are more complicated, in which case the bicubic Coons tends to overshoot the data, i.e. it includes more wiggles than implied by the points. The reason for the overshoot is that the magnitudes of the cross derivatives are much larger than needed. Although the surface is a perfect Coons patch, it may not be a good base surface. A simple solution to this problem is to scale the cross derivatives, if needed. Let $\mathcal{L}_u$ and $\mathcal{L}_v$ be the average arc lengths of the opposite boundary curves in u- and v-directions, respectively. To scale, for example, $\mathbf{D}_1(u)$, let mag be the average magnitude of the cross derivatives, then multiply the curve with a constant factor as follows:

$$\hat{\mathbf{D}}_1(u) = \left(\frac{\mathcal{L}_v}{\mathcal{L}_u + \mathcal{L}_v}\right)\frac{\mathcal{L}_v}{\text{mag}}\mathbf{D}_1(u)$$

This formula first scales the magnitude to become the average arc length, and then it scales again to account for the other directional derivative. After the scaling is done, the bicubic Coons surface will violate the derivative and twist conditions, e.g. the surface's partial derivatives along the boundaries will be different from those of the boundary curves, however, this is irrelevant so long as the base surface is well parametrized and incorporates local details. Fig. 3(a) and (b) shows bicubic base surfaces with derivative scaling. In Fig. 3(a) 30% of the original points were used, and the number of control points were inherited from the boundary curves, i.e. $(n, m) = (20, 8)$. Fig. 3(b) shows the case with 70% of the data points used and $(n, m) = (50, 25)$. Notice that small improvement is gained at a significant cost, i.e. in general it is not worth increasing the number of data points over 30–50%.

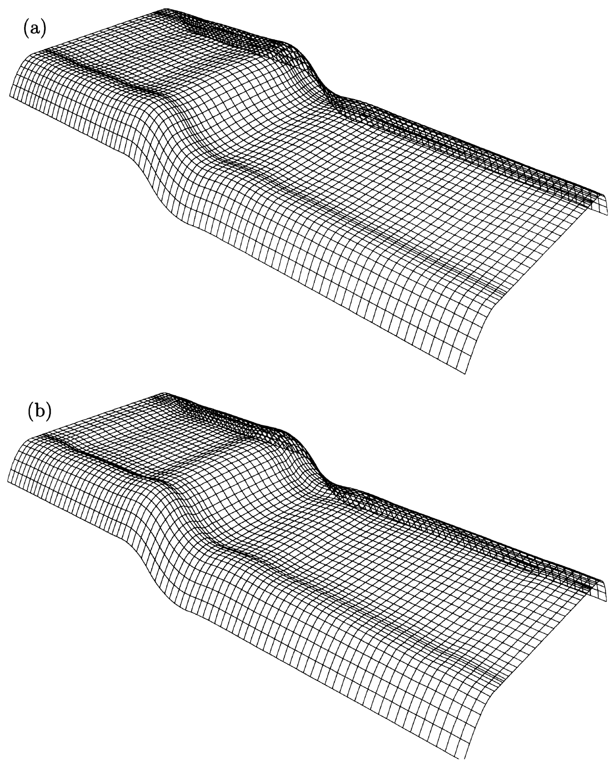


Fig. 3. (a) Bicubic Coons base surface, $(n, m) = (20, 8)$. (b) Bicubic Coons base surface, $(n, m) = (50, 25)$.

2.3. Tensor product

In some cases the parametrization process may require a tensor product surface, i.e. a surface fit is performed to a subset of points given their parameters. The same issues as in the bicubic Coons case apply. That is, we choose a percentage of the original points based on the complexity of the data; the points are randomly picked; the number of control points can be changed; and the parameters are obtained from the bilinear Coons patch. Fig. 4(a) and (b) shows two examples. In the first 30% of the points were used and $(n, m) = (20, 8)$, $(p, q) = (2, 2)$. The second example had the same percentage, however, $(n, m) = (30, 15)$, $(p, q) = (2, 2)$. The large number of data points (3455) and control points (496) required several minutes of compute time (including parametrization and fitting) to obtain the surface shown in Fig. 4(b). Nevertheless, it shows the power behind the tensor product surface fitting. Compare the result of this figure with that of Fig. 3(b). Even though that the Coons patch had many more data points as well as control points, the surface is not as good as the tensor product surface. However, computing the bicubic Coons is much faster because a lot smaller system of equations must be solved.

To conclude the base surface construction, the following is suggested to the reader:

- If the data is simple, use the bilinear Coons patch. It is the absolute fastest and the most reliable method.

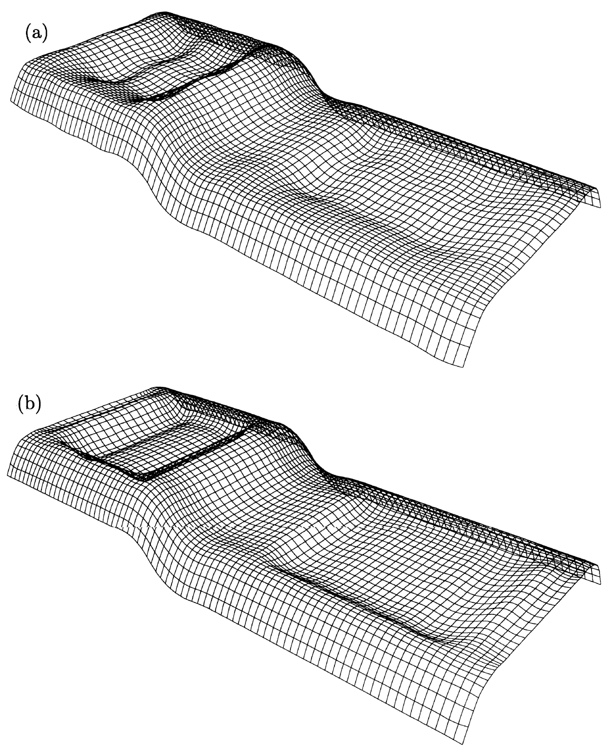


Fig. 4. (a) Tensor product base surface, $(n, m) = (20, 8)$. (b) Tensor product base surface, $(n, m) = (30, 15)$.

- If internal details are to be reproduced, select no more than 30% of the data points to get a bicubic Coons surface.
- If all else fails and intricate details must be reproduced for a good parametrization, use the tensor product surface fitter.

3. Point projection

Once the base surface is computed, the points are projected onto the surface to obtain the parameters. A standard way of doing that is a two step process

1. for each point find a good initial guess; and
2. perform Newton iteration until convergence is reached.

Unfortunately, this is a very expensive and error prone process which can fail quite often especially for points near the boundaries. We propose a different kind of point projection. The idea is that precise Newton iteration is not needed considering the fact that the base surface is quite arbitrary. The method works as follows:

1. decompose the base surface into quadrilaterals using a relative geometric tolerance (we used 0.5% in all test examples);
2. project the points onto the closest quadrilateral; and
3. recover the parameter from the ones of the corners of the quadrilateral.

Details of this method and several speed-up possibilities are explained in the subsequent sections.

3.1. Surface decomposition

Given a (base) surface and a tolerance, a quadrilateral decomposition of the surface is sought so that each quadrilateral does not deviate from the surface more than the tolerance. Such decomposition is normally done adaptively, i.e.

- check if the surface is flat;
- if not, subdivide and recurse.

Although this method works well, it gives rise to cracks in the subdivision, i.e. corners of some quadrilaterals may lie on sides of others. Cracks, in turn, give rise to problems in point projection in that some points may actually project into the crack. For numerically stable point projection we needed a surface decomposition method that gives a crack free blanket covering the entire surface. The following method was implemented:

- Initialize a parameter stack with the parameters of the four corners. Add these parameters to the u- and v-parameter lists, which must be kept in sort order.
- While the stack is not empty do:
  - Pop the stack, i.e. get the parameters of a quadrilateral.
  - Extract a surface patch at these parameters.
  - Check if the patch is flat. If not, then:
    - Compute the midpoint of the parameter span either in u- or in v-direction, depending on which direction to subdivide.
    - If this parameter is not yet in the parameter list, add it to the list keeping the list in sort order.
    - Subdivide the current parameter rectangle into two, and add them to the stack.
- Generate the output blanket by evaluating points at the computed parameters.

A few words about flatness check. Since the parametrization process is quite an expensive task, and since most surfaces in reverse engineering fitting should not be too complex (after segmentation), an inexpensive flatness test was implemented. It works as follows:

- Given a NURBS surface patch.
- Compute an approximating plane to the four corners. This can be a least-squares plane, or a plane passing through the center of gravity of the corners and is parallel to the mid-rulings of the hyperbolic paraboloid determined by the corners. Call this plane the mid-plane.
- Get the normalized implicit equation of the plane.

- Compute the distances of the control points from this plane by plugging their x, y, z coordinates into the plane's equation (this is a cheap way of computing the distance of a point from a plane requiring only three adds and three multiplications).
- If for some point the distance is larger then the tolerance, stop. The surface is not planar and the direction of subdivision is the direction of larger extent.
- Otherwise, the surface is planar.

The direction of subdivision can be obtained more accurately by computing discrete curvatures, however, that would be significantly more expensive. What we needed was (1) geometry based subdivision, (2) reasonable number of quadrilaterals, and (3) very low computational cost. Fig. 5(a) shows a subdivision of the base surface (shown in Fig. 4(b)) using 0.5% relative tolerance. In Fig. 5(b) the same surface is subdivided with 0.1% relative tolerance. Notice that (1) the subdivision is geometry based, i.e. high curvature areas receive more quadrilaterals, and (2) low curvature areas contain more quadrilaterals than necessary due to the crack-preventing nature of the process. This kind of subdivision, although not economical in terms of the number of quadrilaterals, is precisely what we want for obtaining the parameters, i.e. a uniform, geometry/curvature based distribution of quadrilaterals is more important to get good parameters than the minimum number. In fact, the quality of the parametrization can be improved by decreasing the sizes of the quadrilaterals.

3.2. Fast range searching

The next step in the projection process is the projection of points onto the quadrilaterals. A brute force method, i.e. each point is projected onto each quadrilateral to find the closest one, is fairly expensive. For example, the subdivision in Fig. 5(a) has 1080 quadrilaterals, and the data set contains 11,517 points, giving rise to 12,496,680 point projections. One can do significantly better than that.

The idea is to put the points and the quadrilaterals in a data structure for fast range searching. Since range searching in three-dimensional (3-D) is expensive, and since the surfaces are in fact topologically two-dimensional (2-D) entities, point projection to a plane is used. It is assumed that the surface does not curve more than 180° (the point cloud is projectable to a plane), e.g. the method does not work for surfaces such as a complete cylinder or a sphere. The segmentation module must subdivide such surfaces into simpler patches.

The first idea that pops up is to use the well-known least-squares plane. Although it is easy to compute, it tends to behave quite badly especially for thin and long data sets. A better plane is the so-called maximum area plane, i.e. the plane on which the convex hull of the projection forms a maximum area. To obtain such a plane, we use the base surface's decomposition. The derivation below is based upon the ideas and notes of Joe Klahs, which we respectfully acknowledge.

Assume that the base surface is decomposed into quadrilaterals in the form

$$\mathbf{P}_{i,j} \qquad i = 0,\dots \; j = 0,\dots$$

the normal $\mathbf{N}$ of the maximum area plane is sought. Using the diagonals of each quadrilateral in a consistent manner, the dot product

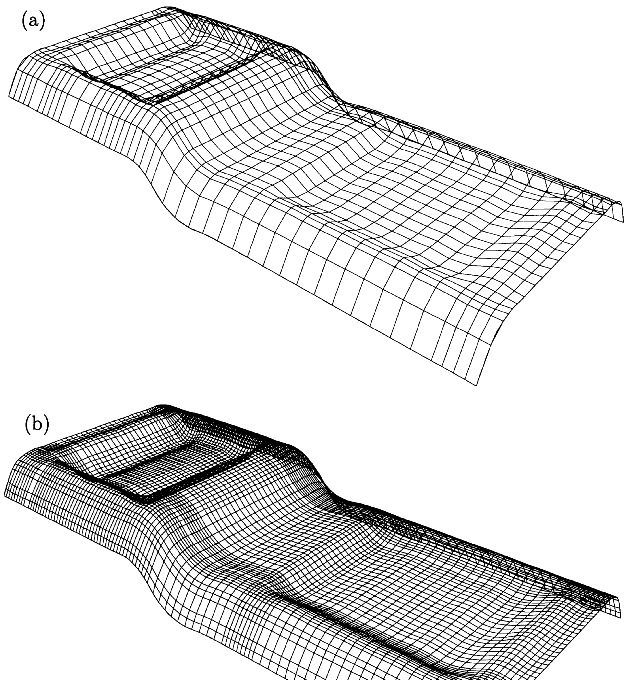
$$\tfrac{1}{2}[(\mathbf{P}_{i+1,j} - \mathbf{P}_{i,j}) \times (\mathbf{P}_{i+1,j+1} - \mathbf{P}_{i,j})]^{\mathrm{T}} \cdot \mathbf{N} = \mathbf{A}_k^{\mathrm{T}} \cdot \mathbf{N}$$

is the area (vector) projected onto the normal (vectors are written as column vectors, i.e. their transpose (T) is needed for product computation). Collecting all triangles, yields the system

$$\mathbf{R} = \begin{bmatrix} \mathbf{A}_1^{\mathrm{T}} \\ \mathbf{A}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{A}_m^{\mathrm{T}} \end{bmatrix} \mathbf{N} = [\mathscr{A}]\mathbf{N}$$

To get the maximum area, one needs the minimum projected area, that is, the minimization problem

$$\min_{\mathbf{N}} \mathbf{R}^{\mathrm{T}}\mathbf{R} \text{ subject to } \mathbf{N}^{\mathrm{T}}\mathbf{N} = 1$$



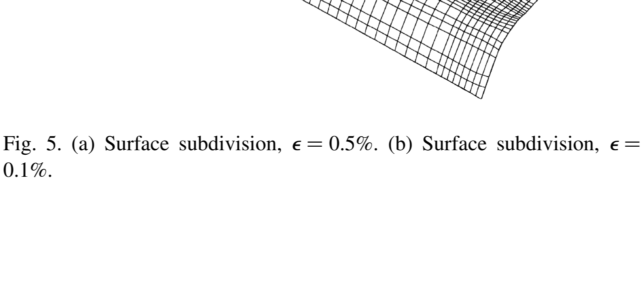


Fig. 5. (a) Surface subdivision, $\epsilon = 0.5\%$. (b) Surface subdivision, $\epsilon = 0.1\%$.

is solved. This leads to the following eigenproblem

$$\mathbf{N}_i^{\mathrm{T}} \mathscr{A}^{\mathrm{T}} \mathscr{A} \mathbf{N}_i = \lambda_i$$

where λ_i are the eigenvalues associated with the eigenvectors $\mathbf{N}_i$, $i = 1, 2, 3$. A quick way of solving this eigenproblem is by iteration

$$\mathbf{N}_{k+1} = \mathscr{A}^{\mathrm{T}} \mathscr{A} \mathbf{N}_k \qquad \mathbf{N}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

If one normalizes $\mathbf{N}_{k+1}$ after each iteration, the iteration above quickly converges to the eigenvector associated with the largest eigenvalue. In all of our practical examples, the iteration converged within five steps using an angular tolerance of 1° (which is more than adequate for practical applications). Fig. 6(a) shows the maximum area plane of the base surface and the projection of the quadrilaterals. The data points and their projections are depicted in Fig. 6(b).

Once the maximum area plane is computed and the points are projected, the plane is put in a position parallel to the $[x, y]$ plane, and the problem is transformed into a 2-D search problem. Fig. 7 shows a 2-D view of the points and the quadrilaterals (the base surface was subdivided using a relative tolerance of 2%). The search proceeds as follows:

- Compute the minmax box $[xl, xr] \times [yb, yt]$ of the projected quadrilaterals and of the points.
- Put a grid over the data set as follows. Let nov be the number of vertices in the surface decomposition, and denote the lengths of the sides of bounding box by x_{L} and y_{L}. Then the size of the grid is

  $$\text{size} = \sqrt{\frac{x_{\mathrm{L}} y_{\mathrm{L}}}{\text{nov}}}$$

  which gives the grid resolutions

  $$x_{\mathrm{r}} = \left\lfloor \frac{x_{\mathrm{L}}}{\text{size}} \right\rfloor \qquad y_{\mathrm{r}} = \left\lfloor \frac{y_{\mathrm{L}}}{\text{size}} \right\rfloor$$

- Bin the projected points into the cells, i.e. for each projected data point (x, y), find the cell indexes

  $$i_{\mathrm{c}} = \left\lfloor \frac{x - xl}{\text{size}} \right\rfloor \qquad j_{\mathrm{c}} = \left\lfloor \frac{y - yb}{\text{size}} \right\rfloor$$

  and register the point with that cell. Note that quadrilateral vertices need not be binned.
- Initialize a distance array for each point with a large number.
- For each quadrilateral in the base surface decomposition do:
  - ∘ Find the cells covering the quadrilateral. This can easily be done by computing the indexes of the cells the vertices fall in.
  - ∘ For each cell do:
    - – Retrieve points registered with the cell.
    - – Project the point onto the quadrilateral.
    - – If the projection falls within the boundaries, and if the distance of the point from the quadrilateral is less than the current distance, then
      - – Compute the parameters from those of the vertices of the quadrilateral.
      - – Update the current distance.

In general each quadrilateral is covered by a few cells, e.g. 4–6, which in turn contain a small portion of the data set. Using the example of Fig. 7, the points are binned into about 160 bins, which in turn contain about 300–350 points per quadrilateral, i.e. only 2–3% of the data set is needed to process a quadrilateral.

3.3. Thinning

In many practical applications data acquisition devices produce much more data than necessary for accurate surface reconstruction, i.e. a subset of the data is quite satisfactory. Such a subset is obtained by the process called thinning. Our method is based on the decomposition of the base surface. The idea is to find the nearest point for each vertex in the quadrilateral decomposition. That is, the decomposition, mentioned in Section 3.1, uses two kinds of tolerances: (1) for point projection; and (2) for thinning. The point projection tolerance (set in the system at 0.5% relative) is hidden from the user, however, the thinning tolerance is an input allowing the user to thin out as many points as necessary. Finding nearest points for each quadrilateral vertex proceeds as follows:

- Use the data structure set up in the previous section.
- For each vertex in the quadrilateral decomposition do:
  - ∘ Find the cell the point is in.
  - ∘ Find the distance between the point and the closest cell wall. Call it the wall distance.
  - ∘ Set the minimum distance to a large number.
  - ∘ While the minimum distance is greater than the wall distance do:
    - – Find the cell ring around the cell of the vertex.
    - – Find the closest point in all the cells in the ring.
    - – Reset the minimum distance to be the distance between the vertex and the closest point.
    - – Increase the wall distance by the size of the grid.

This method first searches the cell the vertex is in. Then it searches rectangular rings of cells around the first cell. The search stops when the distance between the closest point is smaller than the distance between the vertex and the closest wall of the last ring. For most practical applications this algorithm hardly ever goes out of the first ring, i.e. the maximum number of cells it searches is 9. If lots of points are to be thinned out, then considering that the point distribution is much denser than the vertex distribution, only one cell, i.e. the cell the vertex is in, must be searched. Fig. 8(a)

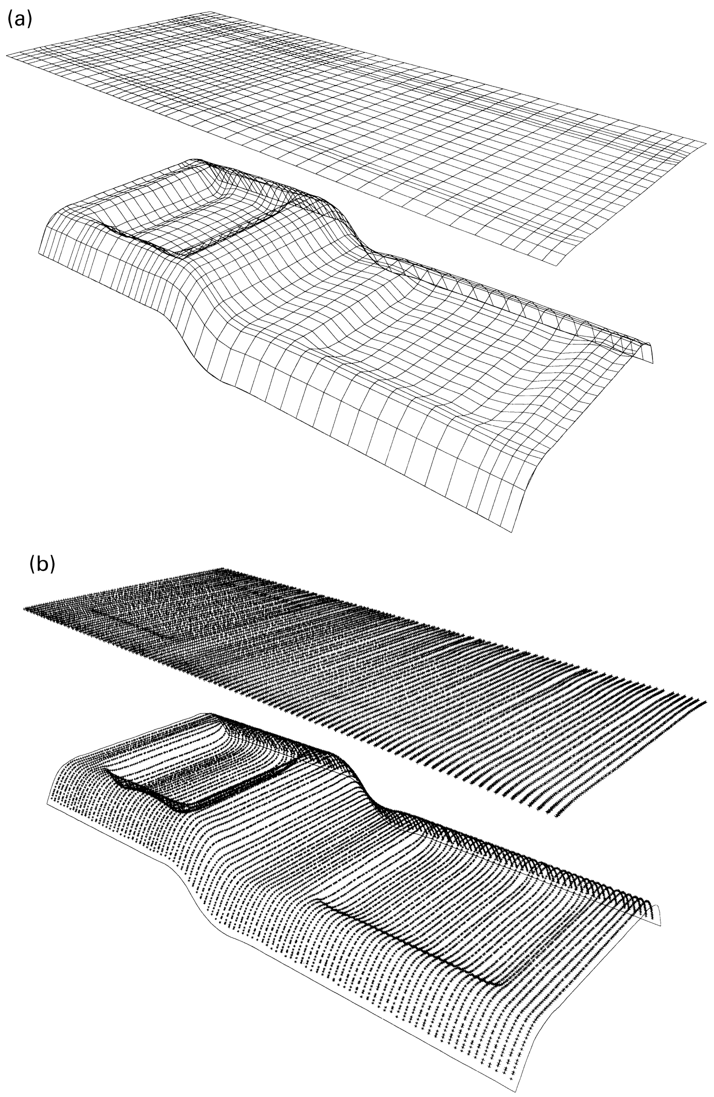


Fig. 6. (a) Projecting the quadrilaterals onto the maximum area plane. (b) Projecting the data points onto the maximum area plane.

shows thinning of the input data using 0.5% relative tolerance. The original 11,517 points have been reduced to a set of only 1052. Fig. 8(b) illustrates thinning with 0.1% tolerance yielding a subset of 4647 points. Notice that the thinning is geometry based, i.e. in flat areas more points are thinned out than in high curvature areas.

3.4. Computing the parameters

The final step in the parametrization process is to compute the parameters from the projections onto the quadrilaterals. If the points have been thinned, then only points in the subset are considered for projection, i.e. thinning is a preprocessing step where selected points are marked. In the projection phase the unmarked points are skipped whereas the marked ones are projected.

A few words about projecting onto quadrilaterals. Four points in space determine a hyperbolic paraboloid, i.e. a bilinear surface. Projecting onto such surface can be done either numerically, e.g. using Newton iterations, or geometrically. Since we introduced surface decomposition to do away with Newton iteration, numerical projection is out of the question. Geometric projection is viable in the following sense. The opposite sides of the quadrilateral (when nonplanar) determine two plane directions. These planes are called the generating planes which can be used to slice the quadrilateral in a parallel manner to obtain its rulings. Putting the planes through the point to be projected, gives two rulings determining a point on the paraboloid which may be considered as the projection. This works quite well for points on or near the paraboloid, and the parameters can be recovered very quickly. However, if the paraboloid is quite flat and the point is far from it, this kind of approximative projection gives fairly bad parameters. Therefore an approximative method was applied, which is justifiable because

- the decomposition is done fine enough (in our experience 0.5% proved to be quite good);

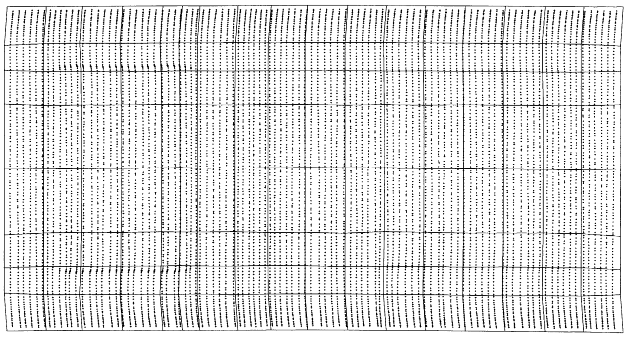


Fig. 7. 2-D view of projected data points and quadrilaterals.

- we are looking for reasonable parameters, not parameters associated with specific points with respect to a given surface, i.e. point inversion.

The method is simple and works as follows:

- Get the mid-plane of the quadrilateral (as above).
- Project the vertices onto this plane. This gives a planar quadrilateral.
- Project the data point onto this plane.
- If the projection falls within the planar quadrilateral, retrieve the parameter based on those of the vertices (this can easily be done based on the two sets of planar rulings emanating from the intersections of the opposite sides).

This process is very fast, reliable, i.e. no numerical computation is needed, and produces good parameters. An added feature is that it can be used for point inversion as well in essentially two ways. The one is to compute good start parameters for fast Newton iteration, and the other is to use a fine surface decomposition for approximate point inversion.

Fig. 9(a) and (b) shows fitting examples using the parameters obtained by the process outlined above. Fig. 9(a) shows a surface fit to 1052 points obtained by thinning out the original data with a tolerance of 0.5%. In Fig. 9(b) the surface was fit to 8061 points, obtained by thinning to 0.05% tolerance; the computational time was quite significant.

4. Conclusions

Several ideas and methodologies for obtaining parameters for surface fitting to random data were presented in this paper. The bottom line of parametrization is to have simple surfaces with as little internal detail as possible. That is, the most critical part of the reverse engineering process is segmentation. Well segmented data sets are easy to thin out, and the thinned set along with the simple boundaries produce high quality surfaces at a reasonable cost. However, if segmentation fails to provide simple patches, and if intricate details are to be reproduced, as in

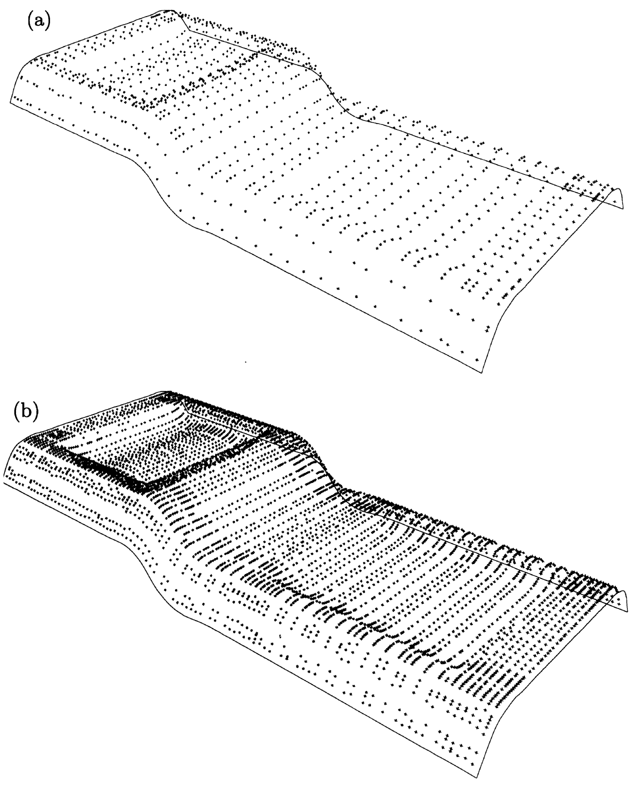


Fig. 8. (a) Data thinning, $\epsilon = 0.5\%$. (b) Data thinning, $\epsilon = 0.1\%$.

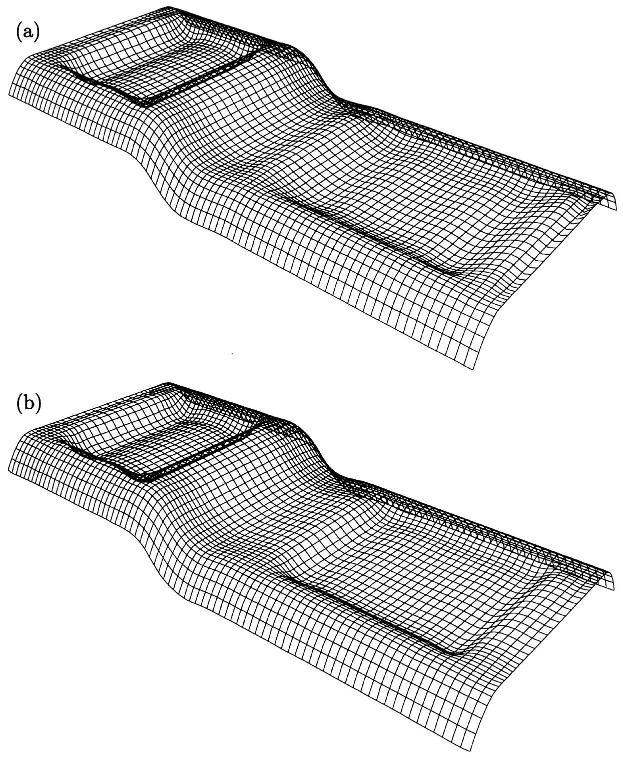


Fig. 9. (a) Final surface fit to 1052 points; $(n,m) = (30,15)$. (b) Final surface fit to 8061 points; $(n,m) = (40,20)$.

the figures of this paper, then the only alternative is to use complicated base surfaces, e.g. bicubic Coons or tensor product surfaces, to interactively adjust their parameters, e.g. number of data points and control points, and to possibly iterate from one base surface to another. That is, the first set of parameters may be used to construct another base surface, which in turn is used to construct a second set of parameters, and so on. There are two major problems with this process: (1) it is extremely time consuming; and (2) if the initial base surface is poor, due to the complexity of the data, then the second one can be even worse, and the entire iterative process produces nonsensical surfaces.

Acknowledgements

The authors thank Prof. C.-H. Menq, Prof. W. Ma, and Dr G. Lukacs for their many valuable comments and criticisms.

References

[1] Amenta N, Bern M, Kamvysselis M. A new Voronoi-based surface reconstruction algorithm, SIGGRAPH 98 Conference Proceedings, 1998. p. 415–21.

[2] Bajaj CL, Bernardini F, Xu G. Automatic reconstruction of surfaces and scalar fields from 3-D scans, SIGGRAPH 95 Conference Proceedings, 1995. p. 109–18.

[3] Bajaj C, Fernardini J, Chen J, Schikore D. Automatic reconstruction of 3-D CAD models. In: Strasser W, Klein R, Rau R, editors. Theory and practice of geometric modeling II Proceedings, Blaubeuren, Germany, 1996.

[4] Benko P, Kos G, Lukacs G, Varady T. Reverse engineering solid objects bounded by regular surfaces, GML 98/4, Computer and Automation Institute, Budapest, 1998.

[5] Benko P, Andor L, Kos G, Varady T. Constrained fitting in reverse engineering, GML 99/4, Computer and Automation Institute, Budapest, 1999.

[6] Chivate PN, Jablokov AG. Solid-model generation from measured point data. Computer-Aided Design 1993;25(9):587–600.

[7] Curless B, Levoy M. A volumetric method for building complex models from range images, SIGGRAPH 96 Conference Proceedings, 1996. p. 303–12.

[8] Eck M, DeRose T, Duchamp T, Hoppe H, Lounsbery M, Stuetzle W. A volumetric method for building complex models from range images, SIGGRAPH 96 Conference Proceedings, 1996. p. 303–12.

[9] Eck M, Hoppe H. Automatic reconstruction of B-spline surfaces of arbitrary topological type, SIGGRAPH 96 Conference Proceedings, 1996. p. 325–34.

[10] Floater MS. Parametrization and smooth approximation of surface triangulation. Computer-Aided Geometric Design XXX;14:231–50.

[11] Guo B. Surface reconstruction: from points to splines. Computer-Aided Design 1997;29(4):269–77.

[12] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized points, SIGGRAPH 92 Conference Proceedings, vol. 26(2), 1992. p. 71–8.

[13] Hoppe H, DeRose T, Duchamp T, Halstead M, Jin H, McDonald J, Schweitzer J, Stuetzle W. Piecewise smooth surface reconstruction, SIGGRAPH 94 Conference Proceedings, 1994, p. 295–302.

[14] Hoppe H. Progressive meshes, SIGGRAPH 96 Conference Proceedings, 1996. p. 99–108.

[15] Hoschek J, Dankwort W, editors. Reverse engineering, B.G. Taubner, Stuttgart, Germany, 1996.

[16] Kos G, Martin RR, Varady T. Methods to recover constant radius rolling ball blends in reverse engineering. Computer-Aided Geometric Design 2000;17(2):127–60.

[17] Lukacs G, Martin RR, Marshall D. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In: Burkhadt H, Neumann B, editors. Computer vision-ECCV, Berlin: Springer, 1998. p. 671–86.

[18] Ma W. NURBS-based CAD modeling from measured points of physical models, Thesis, Katholike Universiteit Leuven, Belgium, 1994.

[19] Ma W, Kruth JP. Mathematical modeling of free-form curves and surfaces from discrete points with NURBS. In: Laurent P-J, Le Méhauté A, Schumaker LL, editors. Curves and surfaces in geometric design, Wellesley, MA: AK Peters, 1994. p. 319–26.

[20] Ma W, Kruth J-P. NURBS curve and surface fitting and interpolation. In: Daehlen M, Lyche T, Schumaker LL, editors. Mathematical methods for curves and surfaces, Nashville, TN: Vanderbilt University Press, 1995. p. 315–22.

[21] Ma W, Kruth JP. Parametrization of randomly measured points for least squares fitting of B-spline curves and surfaces. Computer-Aided Design 1995;27(9):663–75.

[22] Ma W, He P. B-spline surface local updating with unorganized points. Computer-Aided Design 1998;30(11):853–62.

[23] Ma W, Leung PC, Cheung EHM, Lang SYT, Cui H. Smooth multiple surface fitting for reverse engineering. CIRP International Seminar on Manufacturing Systems, Leuven, Belgium, 1999. p. 53–62.

[24] Ma W, Zhao N. Catmull–Clark surface fitting for reverse engineering applications. In: Martin R, Wang W, editors. Geometric Modeling and Processing 2000 Proceedings, Hong Kong, 2000. p. 274–83.

[25] Milroy MJ, Bradley C, Vickers GW, Weir DJ. G^1 continuity of B-spline surface patches in reverse engineering. Computer-Aided Design 1995;27(6):471–8.

[26] Piegl L, Tiller W. The $\mathscr{NURBS}$ book. 2nd ed. New York: Springer, 1997.

[27] Puntambekar NV, Jablokow AG, Sommer HJ. Unified review of 3-D model generation for reverse engineering. Computer Integrated Manufacturing Systems 1994;7(4):259–68.

[28] Sapidis NS, Besl PJ. Direct construction of polynomial surfaces from dense range images through region growing. ACM Transactions on Graphics 1995;14(2):171–200.

[29] Sarkar B, Menq C-H. Smooth-surface approximation and reverse engineering. Computer-Aided Design 1991;23(9):623–8.

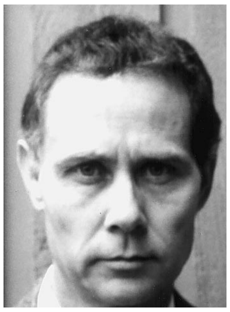
[30] Varady T, Martin RR, Cox J. Reverse engineering of geometric models — an introduction. Computer-Aided Design 1997; 29(4):255–68.

[31] Varady T, Benko P, Kos G. Reverse engineering regular objects: simple segmentation and surface fitting procedures. International Journal of Shape Modeling 1998;4(3/4):127–41.

[32] Varady T, Benko P. Reverse engineering B-rep models from multiple point clouds. In: Martin R, Wang W, editors. Geometric Modeling and Processing 2000 Proceedings, Hong Kong, 2000. p. 3–12.

[33] Weir DJ, Milroy M, Bradley C, Vickers GW. Reverse engineering physical models employing wrap-around B-spline surfaces and quadrics. Proceeings of the Institution of Mechanical Engineers — Part B 1996;210:255–68.

Les A. Piegl is a professor in the Department of Computer Science and Engineering, at the University of South Florida, Tampa, Florida, USA. His research interests are in CAD/CAM, geometric modeling, data structures and alforithms, computer graphics and software engineering. He spent many years researching and implementing NURBS in academia as well as in industry. He is the co-author of the textbook *The NURBS Book*, published by Springer-Verlag. He serves as Editor for *Computer-Aided Design*.



Wayne Tiller is president of GeomWare, Inc., a company specializing in NURBS technology and software. He has 27 years experience in applied mathematics, computer science and software development. He has worked in the area of NURBS geometry since 1981, conducting research and implementing software. He has published numerous papers on this topic and is co-author of *The NURBS Book*. He received a PhD in mathematics from Texas Christian University, USA.