Mathematical techniques for reverse engineering

Michael Floater

April, 1995

1 Introduction

This document describes various mathematical techniques which can be used in constructing geometric models of three-dimensional objects from measured scattered data.

Methods for: detecting feature curves; partitioning the data set accordingly; smoothly approximating the partitions by spline surfaces; and finally building a topologically consistent CAD model are discussed. The overall process is usually referred to as *reverse engineering*.

The main emphasis will be given to scattered data sets from projectable surfaces. An outline of methods for handling other types of data such as cross-sectional data will briefly be reviewed.

2 Background

The purpose of computer-aided design (CAD) is to make the task of designing objects easier. A designer should be able to interactively create, manipulate, and interrogate curves and surfaces on the computer screen in the process of designing an object without needing to actual design it physically [16]. The object could be a car part, a plastic moulding, a turbine blade etc. In CAD the surface of an object is built up from surface patches. A surface patch is typically represented in tensor-product Bézier or B-spline form, in triangular Bézier form, or analytically (e.g. in the case of a sphere). Mathematical techniques such as finite-element analysis can then be applied. The object can often be produced physically by computer-aided manufacture (CAM) via numerically controlled machining.

Yet many designers still prefer to build a physical model rather than use the computer. It is harder to get a 'feel' for a three-dimensional object when it is projected onto the two-dimensional computer-screen. Typically a car-designer of today makes an actual-size sculptured model in clay.

However in order to analyse and manufacture the object it is still necessary to reproduce the object as a CAD model. This leads to the problem of accurately reproducing a physical model in the form of a CAD model. Recently the emergence of highly accurate scanning devices has meant that one can measure a dense sample of points from the surface of the physical model. This has opened up a new possibility; combining mathematical techniques from feature recognition, scattered data approximation, and computer-aided design one can attempt to create the CAD model from the measured data, as far as possible automatically. This process has become known as *reverse engineering* [9]. In practice some intelligent decisions from the user are inevitable yet if the techniques are good enough this user-interaction can be reduced to a minimum.

The following sections contain a collection of basic mathematical steps used in building such a model. Reverse engineering has been discussed previously by Evensen et al. [7,8] mainly from a software engineering point of view. The paper by Evensen and Hull [9] is a shorter overview.

3 Important assumptions on the data

In the following the set of measured data will be indexed by i and the i-th point will be (x_i, y_i, z_i) although it is important to note that no particular ordering on the points is given in advance. The construction of a B-rep model from the data consists of finding one or more piecewise smooth surfaces which approximate the data.

3.1 Functional data

One of the main difficulties with working with scattered data in this form is that one lacks a parametrization or topology. Algorithms for reconstructing surfaces from unorganized points are still in their infancy [13] and are as yet not reliable enough for practical use. We will therefore initially make a practical assumption on the data set. We assume that $\mathbf{s}$, the surface the points are sampled from, can be projected uniquely onto some plane in $\mathbb{R}^3$. In other words we assume that there is some unit vector $\mathbf{e}_3$ such that if (x, y, z) and (x', y', z') are any two distinct points in $\mathbf{s}$ then

$$((x', y', z') - (x, y, z)) \times \mathbf{e}_3 \neq 0. \tag{1}$$

We will say that $\mathbf{s}$ is *projectable*. It follows that $\mathbf{s}$ can be represented parametrically. Indeed by choosing $\mathbf{e}_1$ and $\mathbf{e}_2$ such that $\{\mathbf{e}_i\}$ is an orthonormal set of vectors, it is possible to express $\mathbf{s}$ as

$$(x, y, z) = \mathbf{s}(u, v) = u\mathbf{e}_1 + v\mathbf{e}_2 + f(u, v)\mathbf{e}_3,$$

for some function f. If the $\{\mathbf{e}_i\}$ are known or can be estimated then the problem of approximating $\mathbf{s}$ from the (x_i, y_i, z_i) is reduced to approximating f from the *functional* scattered data $(u_i, v_i, f(u_i, v_i))$. This problem is substantially easier because one can build a topology for the two-dimensional points (u_i, v_i) in the form of a Delauney triangulation [15], for example. For surveys of methods for approximating scattered data by functions see Franke [12] and Schumaker [17].

Typically, even if the surface of the physical object in question is not itself projectable, it can often be decomposed in a natural way into subsurfaces which are projectable at the measuring stage.

One can also consider the possibility of projecting the surface uniquely onto surfaces other than a plane. Such surfaces are called *base surfaces*. For example it may be possible to project the surface uniquely onto a sphere or cylinder. In this case it is appropriate to project each data point onto the sphere or cylinder respectively and use spherical or cylindrical coordinates to represent the projected point. Thus if the base object is represented parametrically as $\mathbf{p}(u, v)$ then the surface would be expressed as

$$\mathbf{s}(u, v) = \mathbf{p}(u, v) + f(u, v)\mathbf{n}(u, v),$$

where $\mathbf{n} = (\mathbf{p}_u \times \mathbf{p}_v)/||\mathbf{p}_u \times \mathbf{p}_v||$, the unit normal to $\mathbf{p}$. Again the problem is reduced to approximating f given functional scattered data $(u_i, v_i, f(u_i, v_i))$.

3.2 Dense data

An obvious necessary assumption on the data set is that the point sample is dense enough with respect to the variation in the surface that a sufficiently accurate model can be generated. Since the surface is unknown, it is impossible to estimate how appropriate the density is from the given data. It is the responsibility of the measurer to ensure that a sufficiently rich point sample is generated.

3.3 Holes

It is of course implicitly assumed that the surface which the data is sampled from is *connected*; a surface is connected by definition. At present, for simplicity, we assume in addition that the surface contains no holes, i.e. it is *simply* connected.

4 Projecting onto a plane

Unless the plane on which the data can be projected 1-1 is known it has to be estimated. This can be done by choosing the least-squares plane through the data set. This in fact implies that the first assumption in Section 3 should be re-defined. The actual assumption is that (1) holds when $\mathbf{e}_3$ is the normal to the least-squares plane through the data points.

A transformation of coordinates is made according to the normal $\mathbf{e}_3$ and the arithmetic mean $\mathbf{a}$ of the points (which lies in this plane). Each data point is shifted and rotated in such a way that $\mathbf{a}$ is shifted to the origin and $\mathbf{e}_3$ is rotated to the unit vector in the direction of the z-axis, i.e. $(0, 0, 1)$. The combination of the two transformations in a solid-body transformation which is a type of affine transformation.

The whole CAD model can be constructed in the new coordinate system using the assumption that the data points are now of the form $(u_i, v_i, f(u_i, v_i))$.

When the final B-rep model is constructed, each spline curve and surface can be transformed back into the original coordinate system. Due to the affine invariance of barycentric combination, each curve or surface is transformed merely by transforming each of its control points [10].

For simplicity of notation we may just as well assume in the remainder of this article that the given data set is functional with respect to the z axis, i.e. we are given points (x_i, y_i, z_i) and there exists some function f such that $z_i = f(x_i, y_i)$. The (unknown) surface we wish to approximate is $(x, y, f(x, y))$.

5 Finding the boundary

The next logical step is to recognise as well as possible the outer boundary of the surface.

We define the boundary to be the broken line interpolant through a subset of the data set, i.e. we identify a closed list of data points whose broken line contains all the data points within the x-y plane. From a topological point of view it is only necessary to consider the x_i and y_i values of each point and construct a two-dimensional boundary. The z_i can be generated afterwards.

One can use an algorithm such as the following. A beneficial side-effect of this algorithm is that the data points are triangulated within the region enclosed by the boundary.

(i) Compute the Delauney triangulation [4] of the (x_i, y_i) points. Its boundary is the convex hull of the data points and acts as an initial guess for the real boundary.

(ii) Choose a parameter $\lambda > 0$.

(iii) Recursively remove any triangle which both (a) contains an edge whose length exceeds λ, and (b) contains an edge belonging to the boundary of the triangulation.

(iv) When no more deletions are possible the algorithm stops and the natural boundary has thus been generated. At the same time, all scattered data points lie on or inside the boundary and they are triangulated.

This simple method of successively eroding the current boundary replaces the initial convex boundary with a boundary which closely hugs the data set; see Figure 1. The user needs to experiment with different values of λ until the boundary is acceptable.

The boundary produced is almost always adequate provided the data points are reasonably distributed. Note that the tolerance λ is absolute. One could introduce a relative tolerance by, for example, scaling λ by the largest edge length in the triangulation. Then $\lambda \in [0, 1]$. The algorithm then removes all edges when $\lambda = 0$ and removes none when $\lambda = 1$.

Sometimes a boundary in the form of point sequences or spline curves may be given in advance, rendering boundary recognition unnecessary.

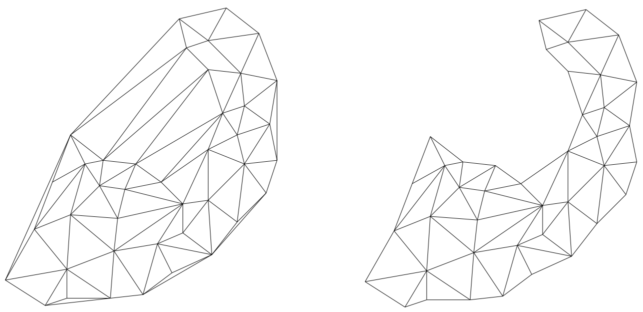

Figure 1: Finding the natural boundary of the data points.

6 Detecting features

Typically the surface will contain *edges* or *creases*, curves along which the cross-derivatives are discontinuous. There may also be curves along which second or higher derivatives in a transversal direction are discontinuous. We call all such curves *feature curves*. The nest task is to detect these feature curves from the data set. This is inevitably partly heuristic and requires user-interaction.

Feature curves are the basis for partitioning the surface naturally into smooth surface patches. The order of continuity between patches will be determined by the type of discontinuity along the feature curves.

Feature detection naturally falls into two stages. In the first we detect feature *points*, points which lie on feature curves and are apparent from the data set due to large jumps in a discrete gradient or higher derivative there. In the second these feature points are joined into sequences. Each sequence of feature points can then be interpolated or approximated by a spline curve, thus yielding an approximate feature curve.

7 Finding feature points

There is a large literature on methods for detecting edges in binary images. In reverse engineering the data and goals are different. Grid data is now scattered data and the detection of jumps in derivatives of the function are more important than jumps in value.

Three possible methods are set out below. After experimentation with real examples, Method 3 has been found to be the most successful among them. Its advantage is that the feature points generated are very accurate provided the data set is relatively noise-free. This is the basis for generating high-quality feature curves.

Method 1. Choose some uniform rectangular grid whose boundary encloses the (x_i, y_i) data points and determine a function value at each node from the piecewise linear interpolant to the data set on the triangulation. Now apply known techniques from image analysis for detecting features.

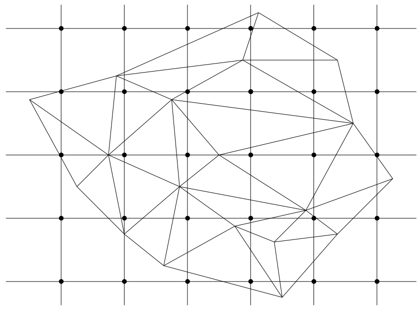

Figure 2: Method 1 — conversion to a uniform grid.

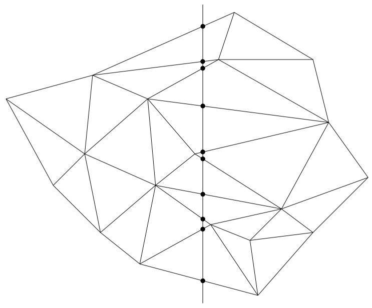

Figure 3: Method 2 — feature analysis along vertical (and horizontal) lines.

Method 2. Again choose some uniform rectangular grid. Now along each line of the grid, find all its intersections with edges of the triangulation. Estimate a function value at each intersection by linearly interpolating the two function values at the ends of the line segment. Now apply a *univariate* feature analysis along the line, identifying all occurrences of jumps in gradient and second derivative that exceed some threshold.

Method 3. Extend an infinite line through each *edge* **e** of the triangulation. Compute a function value at each of the intersections of the line with other edges of the triangulation using linear interpolation as in Method 2. The two function values at the ends of edge **e** are of course given values. Then, as in Method 2, carry out a univariate feature analysis along this line, but only considering the existence of a feature point lying along edge **e**. This means that it is only necessary to find a window of function values along the line, in each direction. Typically only a few points in each direction are sufficient.

The image analysis in Method 1 can be done via spline decomposition. However the drawback of this method is that when the scattered data is approximated by grid data, a lot of information and features can be lost if the grid density is low. Conversely new features, namely the edges of the triangulation may suddenly appear if the density of the grid is high. Since, in general, the distribution of scattered data varies in the domain it is impossible to resolve

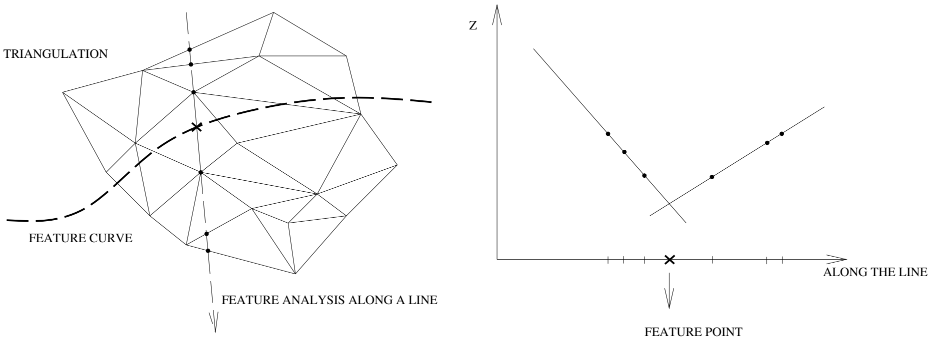

Figure 4: Method 3 — feature analysis along the extended line through each edge

these conflicting problems. Probably for these reasons, Method 3 has usually performed best. Method 2 can be regarded as a hybrid of the other two.

The advantage of Method 3 over the other two is that as little as possible data is thrown away or approximated. A disadvantage is that known techniques from image analysis are no longer relevant and so one has to develop new algorithms.

8 Finding feature points in a univariate function

Feature detection has now been reduced to determining the existence of a feature point in a given interval of a function $\mathbb{R} \to \mathbb{R}$. In this section a brief summary is given of some of the basic ideas which can be used for this. The reader is referred to Chen and Schmitt [3] for some references in the area of feature analysis of uniformly spaced range data. Some of the following methods are similar to theirs.

Divided differences and B-splines. Detecting features essentially requires analysing divided differences. Since we are working with non-unform data (unlike in image analysis) this involves non-uniform divided differences. The function $[t_i, \ldots, t_{i+k}]y$ is a k-th order divided difference of a function f. For example

$$[t_0]f = f(t_0), \quad [t_0, t_1]f = \frac{f(t_1) - f(t_0)}{t_1 - t_0},$$

and

$$[t_0, t_1, t_2]f = \frac{[t_1, t_2]f - [t_0, t_1]f}{t_2 - t_0} = \left(\frac{y_2 - y_1}{t_2 - t_1} - \frac{y_1 - y_0}{t_1 - t_0} \right) / (t_2 - t_0).$$

In general

$$[t_i, \ldots, t_k]f = ([t_{i+1}, \ldots, t_k]f - [t_i, \ldots, t_{k-1}]f)/(t_k - t_i),$$

see de Boor [5] for more details.

An interesting observation is that B-splines occur when one takes divided differences of piecewise polynomials. A function having a discontinuity in some derivative can be approximated by a piecewise polynomial and this means that feature analysis can be aided by some knowledge of B-splines. Of particular importance is the identity [5]

$$N_{i,k,\mathbf{t}}(t) = (t_{i+k} - t_i)[t_i, \ldots, t_{i+k}](\cdot - t)_+^{k-1}.$$

Here, $N_{i,k,\mathbf{t}}$ is a B-spline of order k (or degree $k-1$) defined on the knot vector $\mathbf{t} = \ldots, t_i, t_{i+1}, \ldots, t_{i+k}, \ldots$. Given a function $f : \mathbb{R} \to \mathbb{R}$, f_+ is defined as $f_+(x) = f(x)$ for $x \geq 0$ and $f_+(x) = 0$ otherwise.

The basis for finding jumps in first and second derivative can thus be based on properties of B-splines. If it has been decided that there is a feature point in the interval in question, its location and z value can be estimated using Lagrange interpolation [2] for the points on either side of the interval.

The location of the feature point found in this way is often very accurate provided the data is relatively noise-free. The feature points generated by many feature-detection algorithms used in image analysis are forced to be pixels, i.e lie on the underlying grid of nodes. To get good quality it is clearly important to be able to find the proper location of the feature point which is usually somewhere between nodes.

A geometric approach. Instead of detecting jumps in first and second derivative, one could detect jumps in their geometric analogues: angle and curvature. Essentially, the same basic ideas are still valid. Curvature can be regarded as a divided difference of angle and angle a divided difference of $\theta(\mathbf{x})$ where $\mathbf{x} = \cos(\theta)x + \sin(\theta)y$.

This geometric alternative is perhaps easier for the user to understand but somewhat more complicated to implement. The thresholds should correspond to the minimum change in angle and minimum change in curvature.

In summary the following feature point types can be detected.

(1) jumps in first derivative (where g is not C^1),

(2) jumps in second derivative (where g is not C^2),

(3) jumps in angle (where g is not G^1),

(4) jumps in curvature (where g is not G^2),

After extensive testing it has been found that, once the tolerances were matched correctly, the set of feature points of type 1 found by (1) and (3) are almost identical. Similarly (2) and (4) produce almost entirely the same type 2 feature points.

Futher useful information. When we find C^1 or G^1 feature points, we can store in addition the sign of the jump in first derivative or angle respectively. If the sign is positive, we set the type to be 1, and if negative, -1. Later when we construct feature curves from the feature points (in the full scattered data model), the two types of point (1 and -1) are treated separately.

9 Generating feature curves from feature points

We now have a set of feature points. The next task is to build sequences of feature points which can be approximated by spline curves, yielding feature curves. In fact each feature points has a type, C^1 or C^2. And if it is of type C^1 we further classify it according to whether the curvature is positive or negative. It is clear that each feature curve will consist of feature points of constant type. We thus apply the sequence building to each group of points separately.

The first step is to clean out outlying points. Feature points corresponding to a single feature curve may also appear as a band of points. These need to be thinned out. In image analysis algorithms are known for generating the so-called skeleton of a subset of nodes of a regular rectangular grid. It is possible to adapt this kind of 'thinning' algorithm to work with nodes of a triangulation.

Having thinned out the feature points sufficiently, one can triangulate them. The user supplies a distance threshold d and angle threshold α. We then begin at an arbitrary point, and according to d and α, build a list of feature points such that each consecutive pair form the end points of an edge of the triangulation and such that the length of each of these distances is less than the d and the change in angle between each pair of edges is less than α. Another criterion for choosing the next point in the sequence is curvature. For a discussion of list-building of this kind, see Dedieu and Favardin [6]. A third parameter k, an integer, is used to decide when a sequence is so short that it can be regarded as noise and discarded. Typically using $k = 3, 4, 5$, a lot of noise is thrown away.

When this is finished, the result is a collection of lists of feature points of uniform type. Each list can now be converted to a B-spline curve.

If the feature points are very accurate (for example if the features in the surface are very clean and the sample of data is very dense), it may be desirable to *interpolate* them with a spline curve. Otherwise it is better to *approximate* them in order to smooth out unwanted variations.

10 Building a topology

The task now is to build a planar graph based on the feature curves where each arc is a spline curve. This graph must be topologically consistent, i.e. the end of every curve must meet at least two other such curves. Initially this will not be the case. Each feature curve will often be fragmented. Usually feature curves will also end before they reach an intersection point.

Thus it is vital to have a clean-up operation. Gaps in feature curves are identified by the user and are then automatically filled in. The user can also interactively drag the end point of a curve towards a desired intersection point. Using elementary operations of this kind a consistent graph of curves is constructed. The outer boundary curves of the graph are approximations to the points belonging to the boundary found earlier in Section 5.

When the graph is complete, it may be desirable to add further curves in order to split some of the faces into smaller ones. This may be necessary so that

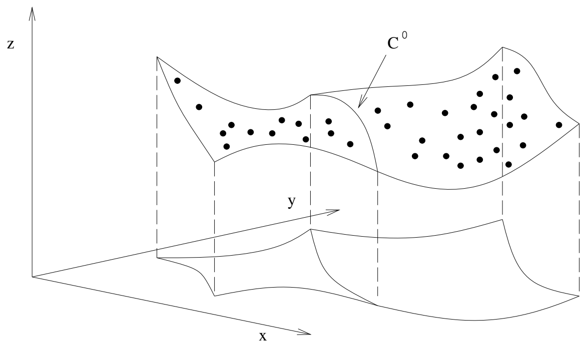

Figure 5: Two adjacent faces of the graph.

all faces have either three or four boundary curves.

11 Surface construction

The final phase of the reverse engineering is to approximate the data points and feature curves by a network of smooth surface patches which join together with appropriate orders of (geometric) continuity.

One surface patch will be generated for each face in the planar graph. The resulting collection of matching surface patches together with the topological structure will constitute a B-rep model; see Figure 5.

For the sake of simplicity we will first of all describe one possible method for generating the surface patches in such a way that the entire surface is C^0. In order to achieve this it is clear that we can work with each set of boundary curves and the subset of scattered data contained by them (in the x-y plane) in isolation provided that the patch boundaries interpolate the boundary curves exactly. The subset of the data set lying inside the planar boundary curves is determined by blending the boundary curves with a two-dimensional surface. This can be a Coons blend and will map $\mathbb{R}^2$ to $\mathbb{R}^2$. Then a test is carried out for each (x, y) data point in turn to see whether it intersects this blend surface. The implicit assumption made here is that the blend surface lies entirely inside the boundary curves it blends. In practice this is almost always the case since the boundary curves do not normally oscillate greatly. If they do they should be split into smaller but smoother curves and the face of the graph split accordingly. Thus we have reduced the surface generation to the following problem.

Statement of the problem. Given a set of functional spline curve boundaries and a sample of functional scattered data lying inside them, construct a parametric surface which simultaneously

(1) blends (interpolates) the boundaries,

(2) approximates the internal scattered data.

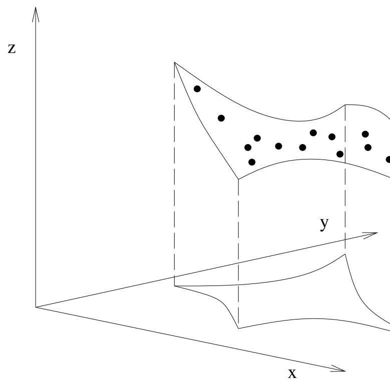

Figure 6: The boundary curves and the internal scattered data.

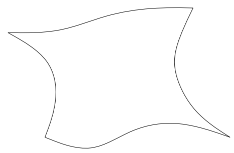

Figure 7: A four sided patch.

Boundary curves will typically be C^2 cubic splines; see Figure 6.

The easiest case to deal with is when the face has four edges for then it is natural to construct a tensor-product spline surface; Figure 7. However if there are three or even two boundary curves, the surface patch can still be represented as a tensor-product spline surface by regarding one or two of the sides as degenerate; see Figure 8. If there are more than four sides, as in Figure 9, the region can be subdivided in such a way that each subregion does have four sides.

Method for surface generation. This is a multiphase solution.

1. First blend the two-dimensional boundaries with the tensor-product spline surface $\mathbf{s}_1 : [0,1] \times [0,1] \to \mathbb{R}^2$. We have used the Coons blend mentioned above. The first two coordinates (x, y) of the three-dimensional surface $\mathbf{s}_2$ to be built are precisely this blend surface. If necessary insert extra knots into $\mathbf{s}_1$ in order

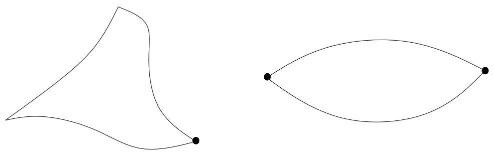

Figure 8: Two and three-sided patches.

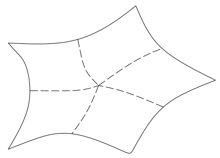

Figure 9: Subdividing a five-sided patch.

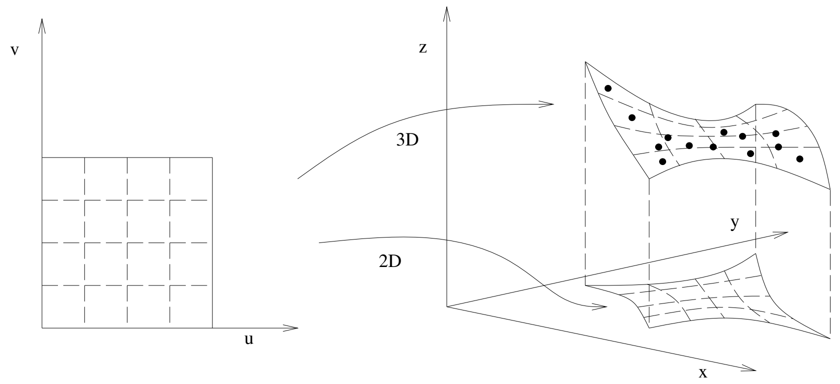

Figure 10: The blend surface makes a parametrization.

to increase the number of degrees of freedom in $\mathbf{s}_2$ and hence the accuracy of the final approximation.

3. Project each scattered data point (x_i, y_i, z_i) onto $\mathbf{s}_1$ to find the corresponding (u_i, v_i) parameters. This is achieved by intersecting $\mathbf{s}_1$ with the point (x_i, y_i). Thus a *parametrization* is obtained for the data points. The parametrization will certainly not be optimal. It will be especially poor when the surface has large gradients with respect to the x-y plane. Possible improvements will be mentioned in Section 12.

4. Form a new scattered data set consisting of the points in the form (u_i, v_i) with z-values z_i plus a sample of points along the boundaries. This is now functional scattered data lying in the unit square and can be approximated in various ways. We wish to generate a spline spline surface so one could use the so-called 'mask-method', described in Arge, Dæhlen, Tveito [1], to generate a uniform mesh of function values. These can be interpolated by, for example, C^2 bicubic tensor-product spline $\mathbf{f} : [0, 1] \times [0, 1] \to \mathbb{R}$. However this method can produce unwanted oscillations and other two-phase methods have been experimented with, in particular using piecewise polynomials on a triangulation.

5. Evaluate $\mathbf{f}$ at the Schoenberg points w.r.t. the knot vectors of the blend surface $\mathbf{s}_1$. These are interpolated to provide the internal control points of $\mathbf{s}_2$. Control points lying along the boundaries are copied directly from the control points of the boundary curves to ensure that $\mathbf{s}_2$ exactly interpolates the boundary curves.

12 Future work

A major research aim at present is to handle scattered data which is not projectable. In a recent research project, Geir Westgaard and the author have constructed surface triangulations from cross-sectional data [11]. The author has also investigated ways of approximating surface triangulations or regions of them by tensor-product splines.

Triangulations from cross-sections.

Given a sequence of parallel cross-sections, each containing contours, we have constructed a smooth triangulation approximating the data [11]. The basic philosophy is to choose a suitable bounding region $\Omega \subset \mathbb{R}^3$, and build a smooth function $f : \Omega \to \mathbb{R}$, having the contours as zeros. The whole zero set of f is then a smooth surface (or manifold) approximation to the data. The zero set can be approximated by a triangulation by evaluating f at every node of a cubical lattice and using any of a variety of known techniques in computer graphics, such as the 'marching cubes' algorithm, for triangulating the isosurface.

In particular, if each slice is normal to the z axis, one can, making some reasonable assumptions, solve the biharmonic equation in (x, y) in each slice with appropriate boundary conditions and apply a cubic spline approximation in z to compute f. This yields a C^1 surface approximation without needing a great deal of computing time. In principle f and its zero set can be made to be C^k.

This implicit method has many advantages over traditional (interpolatory) triangulations for cross-sections [18,19]. The main one is that the problem of handling objects with branches is automatically solved in a natural way. Another is that one can smoothly interpolate between slices.

The idea of using an implicit method for cross-sectional data was presented recently by Jones and Chen [14], but has also been suggested by others [18]. Implicit methods have also been used recently for tackling arbitrary scattered data from surfaces by Hoppe, DeRose et al. [13]. The main emphasis of the methods presented in [11] is on making the surfaces as smooth as possible with a view to their potential use in reverse engineering.

Tensor-product spline approximation.

A second area of interest is looking for better ways of approximating scattered data within a boundary with tensor-product splines. Whether or not the data is functional, it is still necessary to use a parametric surface (Section 11) rather than a function to interpolate the boundary curves since it is unlikely that the boundary is a rectangle.

When the surface is projectable, the obvious method for making a parametrization as in Step 3, in Section 11 is projection. But this can yield a rather skew parametrization. If the gradient of the function is large in places, any scattered data approximation is likely to exhibit oscillations. A prototype of a method for making a more natural parametrization is currently be developed. This will also be relevant for approximating non-projectable data sets.

There are numerous known ways of parametrizing points on a curve. The two most popular are probably uniform and chord length. We should be aiming to generate analogous parametrizations for points on a surface.

References

[1] Arge E., M. Dæhlen, A. Tveito, A strategy for the approximation of large scattered data sets, J. Comp. Appl. Math. **59** (1995), 191–205.

[2] Atkinson K., *An Introduction to Numerical Analysis*, Wiley, New York, 1989.

[3] Chen X., Schmitt, F., Surface modelling of range data by constrained triangulation, Comput. Aided Design **26** (1994), 632–645.

[4] Cline A. K., R. L. Renka, A storage-efficient method for construction of a Thiessen triangulation, Rocky Mount. J. Math. **14**

[5] de Boor C., *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[6] Dedieu J. P., Ch. Favardin, How to draw a curve using geometrical data, in *Curves and Surfaces, Laurent, Méhauté, and Schumaker (eds.), Academic Press, Boston* (1991), 135–138.

[7] Evensen P., M. Irgens, S. Carlsen, A. Ytrehus, M. Dæhlen, GeoRes, SI Report No. 90539-1, 1991.

[8] Evensen, P., S. Carlsen, G. Westgaard, Utviklingsprosjekt MNS — GeoRes, SI Report No. 910810-1, 1991.

[9] Evensen, P., S. Hull, From prototype to CAD model — A method for Reverse Engineering, SINTEF-SI Report No. STF33 A93014, 1993.

[10] Farin G., *Curves and surfaces for computer aided geometric design*, Academic Press, San Diego, 1988.

[11] Floater M. S., G. Westgaard, Smooth surface reconstruction from cross-sections, SINTEF Report, 1995.

[12] Franke R., Recent advances in the approximation of surfaces from scattered data, in *Topics in Multivariate Approximation*, C. Chui, L. Schumaker, F. Uteras, (eds.), Academic Press, New York (1987) 79–97.

[13] Hoppe H., T. DeRose, T. DuChamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, Computer Graphics, Vo. 26, No. 2 (1992), 71–78.

[14] Jones M. W., M. Chen, A new approach to the construction of surfaces from contour data, Computer Graphics forum, Vol. 13, No. 3 (1994), 75–84.

[15] Preparata, P., M. Shamos, *Computational geometry, Springer-Verlag*, New York, 1995.

[16] Sabin M., Numerical geometry of surfaces, Acta Numerica (1994), 411–466.

[17] Schumaker L. L., Fitting surfaces to scattered data, in *Approximation Theory II, G. G. Lorentz, C. K. Chui & L. L. Schumaker (eds.), Academic Press, New York* (1976), 203–268.

[18] Schumaker L. L., Reconstruction of 3D Objects from cross sections, in *Computation of Curves and Surfaces, W. Dahmen, M. Gasca, and C. Micchelli eds., Kluwer, Dordrecht* (1990), 275–309.

[19] Schumaker L. L., Triangulations in CAGD, IEEE Computer Graphics and Applications (1993), 47–52.