Title
Modeling Contours of Trivariate Data

Journal
Mathematical Modelling and Numerical Analysis (Modelisation Mathematique et Analysis Numerique), 26

Author
Hamann, Bernd

Publication Date
1992

Peer reviewed

MATHEMATICAL MODELLING AND NUMERICAL ANALYSIS
MODÉLISATION MATHÉMATIQUE ET ANALYSE NUMÉRIQUE
(Vol. 26, n° 1, 1992, p. 51 à 75)

MODELING CONTOURS OF TRIVARIATE DATA

by B. HAMANN ([1])

Abstract. — *A general scheme for computing contours of trivariate data is discussed. It is assumed that three-dimensional points with associated function values are given without any other information. The goal is to construct a smooth approximation to a contour of these data. Usually, an interpolating or approximating function is constructed in order to estimate values on a whole three-dimensional domain. Very often, the resulting function is represented by a set of contours which are surfaces in space. Here, a method is described that first estimates points on a particular contour, generates a piecewise linear approximation to that contour, and finally uses this linear approximation as input for a surface scheme. The surface scheme then yields a surface which approximates the desired contour. Applications for this technique are found in medicine (Computerized Tomography* (CT), *Magnetic Resonance Imaging* (MRI)), *meteorology (temperature measurements) and physics in general. Particularly in medical applications one is more interested in contours and the shape of objects than in a function that interpolates measurements.*

Keywords : *Contour, curvature, data reduction, G^1 surface, topology, triangulation, trivariate data.*

Résumé. — Modélisation de lignes de niveaux pour des données à trois variables. *Un schéma général pour le calcul d'iso-contours de données à trois variables est examiné. On suppose que seuls les points tridimensionnels et la valeur de la fonction en ces points sont donnés. Le but est de construire une approximation régulière pour un iso-contour de ces données. D'habitude, une fonction d'interpolation ou d'approximation est construite pour en déduire les valeurs sur un domaine tridimensionnel complet. Souvent la fonction ainsi obtenue est représentée par un ensemble d'iso-contours qui sont des surfaces dans l'espace. Ici une méthode est décrite qui estime d'abord les points sur un iso-contour donné, génère une approximation linéaire par morceau de ce contour, et utilise cette approximation pour un schéma de génération de surface. Ce schéma donne une surface qui approxime l'iso-contour choisi. On trouve des applications de cette technique en Tomographie par ordinateur* (CT), *Visualisation par Résonance Magnétique* (MRI), *météorologie (mesure de températures) et en physique en général. Plus précisément dans les applications médicales, on s'intéresse plus à des iso-contours et à la forme d'objets qu'à des fonctions interpolant des mesures.*

([1]) Department of Computer Science, Mississippi State University, Drawer CS, Mississippi State, MS 39762, U.S.A.

M² AN Modélisation mathématique et Analyse numérique 0764-583X/92/01/51/25/$ 4.50

INTRODUCTION

In the following, it will be assumed that the given data are either of the form

$$\{(\mathbf{x}_i, f_i) = (x_i, y_i, z_i, f_i) \mid \mathbf{x}_i \in \mathbb{R}^3, f_i \in \mathbb{R}, i = 1 \dots n\} \quad (1)$$

or

$$\{(\mathbf{x}_i, f_i) = (x_{i,j,k}, y_{i,j,k}, z_{i,j,k}, f_{i,j,k}) \mid \mathbf{x}_i \in \mathbb{R}^3, f_i \in \mathbb{R}, i = 0 \dots n_i, j = 0 \dots n_j, k = 0 \dots n_k\} . \quad (2)$$

In the first case, the data are given scattered in three dimensional space, in the second case they are organized in a rectilinear fashion. Therefore, the terms scattered data and rectilinear data will be used in the following. Figure 1 shows the two different data types. The desired goal is to compute an approximation to one or more contours of the data. A contour is defined as the set of all points for which f = const holds, when f is the (unknown) discretized function.

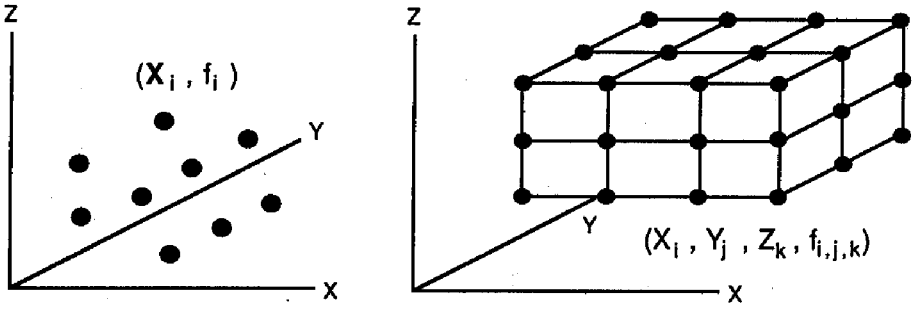

Figure 1. — Scattered and rectilinear data in three-dimensional space.

There exist two basic approaches to deal with these data : either one uses volume visualization techniques, if one is interested in rendering the original data only, or one uses numerical methods, if one wants to obtain a mathematical description of a trivariate approximating function to the data or an approximation of certain contours of the data. Most volume visualization methods are based on ray tracing techniques. A finite number of rays (equal to the number of points to be colored on a screen) intersecting the domain of the scattered or rectilinear data is used to determine color values when the data are to be rendered directly. These methods can be quite expensive, when large numbers of rays have to be considered (usually $1{,}024^2$ rays per image). Effects like transparency of volumes can be simulated.

In this paper, it will not be explained which techniques could be used to visualize this kind of given data directly using volume visualization. Appropriate algorithms are given in the standard computer graphics literature, e.g., Drebin *et al.* [13], Foley *et al.* [13], Fuchs *et al.* [22], Hamann [24], Kajiya/von Herzen [27], Levoy [30], [31], Ney *et al.* [34], Nielson/Hamann [36], Sabella [41] and Tiede *et al.* [44]. The general problem of data acquisition and noise reduction in the case of physical measurements is discussed in the field of computer vision and pattern recognition (see Ballard/Brown [9] and Fu *et al.* [22]).

One common approach in computer-aided geometric design to model this kind of data can be divided into three steps : first, certain derivative estimates at the data points are generated, second, an approximating trivariate function is constructed and third, the result is rendered, usually as a set of contours. Stead [43] and Zucker/Hummel [46] give methods for estimating gradients for this kind of data, scattered data interpolation methods are described in Alfeld [1], Barnhill [6], Bloomquist [7], Franke/Nielson [20], Hoschek/Lasser [26], Petersen *et al.* [37] and Worsey/Farin [45]. Again, these techniques will not be covered here.

The whole modeling process to be explained here can be divided into the following sequence of steps :

(i) A method for data reduction will be given as a modification and extension of a bivariate point removal procedure (see Le Méhauté/Lafranche [29]).

(ii) Algorithms will be presented for obtaining points on a contour considering only the given scattered or rectilinear data or a reduced data set as the result of step (i) in scattered form. The algorithms will be different for scattered and rectilinear data. Triangles will be constructed from these contour points as a piecewise linear approximation to a contour. They will be obtained by triangulating closed nonplanar polygons in three-dimensional space. In Choi *et al.* [11] a criterion is proposed for a '' good '' triangulation in space. In Lorensen/Cline [33] contours are generated for rectilinear data.

(iii) Further, it will be explained how to get topological information, e.g., the neighbor triangles of a triangle and the component of a contour a particular triangle belongs to (a contour can be separated into multiple components).

(iv) A surface will be computed that interpolates to all the points approximating the contour and eventually to prescribed normal vectors at those points (see Barnhill *et al.* [5], Boehm *et al.* [8], Farin [16], [17], Hagen/Pottmann [23], Hamann *et al.* [25], Nielson [35] and Pottmann [39] for more details on surfaces).

(v) Finally, a technique will be introduced for estimating the curvature behavior of a surface, if one knows a triangulated version of the surface only. This technique can then be used for interrogating the " smoothness " of the surface created in step (iv).

DATA REDUCTION

Data reduction should be done first among the five modeling steps. The paradigm for data reduction is quite easy : if a subset of the given function values can be approximated locally by a linear (trivariate) polynomial within a prescribed tolerance, certain data points associated with the approximated function values in this subset will be removed. They can be considered not being significant, because a contour for the data will later also be approximated by using a linear polynomial scheme. What is meant by the term " subset " will be explained below. The data reduction procedure presented here is a modification and extension for trivariate data of the algorithm described in the publication by Le Méhauté/Lafranche [29]. For this point removal process, the data can be given in scattered or rectilinear form. For the further discussion it is more convenient to ignore the implied structure of rectilinear data.

First, the given point set $\{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbb{R}^3, i = 1 \ldots n\}$ with associated function values $\{f_i \mid f_i \in \mathbb{R}, i = 1 \ldots n\}$ will be triangulated. It is assumed that the resulting triangulation is the Delaunay triangulation D_0 of the point set. Whenever the term triangulation is used in combination with this point removal procedure (and only in this context), one actually is dealing with tetrahedra in three-dimensional space. Given the triangulation of the data points and the function values f_i associated with them it is possible to construct a C^0 piecewise polynomial interpolating trivariate function for the data. This can easily be done by defining the function value at a point $\mathbf{x}$ in a tetrahedron as $f(\mathbf{x}) = u_1 f_1 + u_2 f_2 + u_3 f_3 + u_4 f_4$, where (u_1, u_2, u_3, u_4) are the barycentric coordinates of the point $\mathbf{x}$ relative to the vertices of the tetrahedron it is lying in, and f_1, f_2, f_3 and f_4 are the known function values at these vertices. For the point removal process, it is not necessary to achieve higher order continuity for the interpolant.

The basic idea of the point removal step is to iteratively replace the four vertices of an interior tetrahedron by one point and retriangulate the data set locally (" interior " will be explained below). Each interior tetrahedron will be assigned a weight which is a measure for its significance to the implied C^0 interpolant. Data points are removed only if they are lying inside the convex hull of the original point set such that the domain of the interpolant still remains the same after point removal. Tetrahedra will be removed if the interpolant for the reduced data set does not differ more than

a prescribed tolerance ε from the interpolant for the original data set. To describe the procedure more formally the following notation will be used :

- V_t is the set of the four vertices $\mathbf{v}_i$, $i = 1 \ldots 4$, constituting a particular tetrahedron t.
- T_t is the set of all tetrahedra having either one, two, or three vertices in common with tetrahedron t.
- P_t is the set of all vertices lying on the polyhedral boundary of the region implied by tetrahedron t and all the tetrahedra in T_t.
- $pred_t$ is a predicate for tetrahedron t which is *true* if all the points in V_t are lying inside the convex hull of the original point set (tetrahedron t is called an " interior " tetrahedron) and all tetrahedra in T_t are tetrahedra in the Delaunay triangulation D_0.

The predicate for a tetrahedron will later ensure that only convex regions have to be retriangulated and that the significance of a tetrahedron is always measured with respect to the original triangulation D_0 and not with respect to an already modified triangulation. If a tetrahedron's predicate is *true* its weight ω_t can be computed in the following way :

1. Compute the centroid $\mathbf{c} = \frac{1}{4} \sum_{i=1}^{4} \mathbf{v}_i$ of the vertices of tetrahedron t and assign the function value $f(\mathbf{c}) = \frac{1}{4} \sum_{i=1}^{4} f_i$ to it (average of the given function values at the four vertices).
2. Compute the (locally) new triangulation $\bar{T}$ of the point set $P_t \cup \{\mathbf{c}\}$ by connecting each point in P_t with $\mathbf{c}$. Considering the fact that P_t describes a convex polyhedron, this way of retriangulating is one possible way of doing it.
3. Compute the (local) difference of the new piecewise linear spline S_1 based on the reduced point set and the previous piecewise linear spline S_0 based on the unreduced point set. S_0 and S_1 differ only on the convex region implied by the polyhedron with vertices in P_t. Therefore, one has to consider the local triangulation $\bar{T}$ only and the difference between the two splines S_0 and S_1 is given by

$$\| S_0 - S_1 \| = \sum_{\bar{t} \in \bar{T}} \| S_0 - S_1 \| \big|_{\bar{t}} . \tag{4}$$

The norm that is used here to measure the difference is a very simple discrete norm, $\| \ \|_{dis}$, defined as

$$\| S_0 - S_1 \|_{dis} = \sum_{i=1}^{4} | f_i - S_1(\mathbf{v}_i) | . \tag{5}$$

It is a norm on the set of all polynomials of degree ≤ 1. The weight for a tetrahedron t is defined as

$$\omega_t = \| S_0 - S_1 \|_{dis} . \tag{6}$$

To compute ω_t one has to determine the tetrahedra in the new triangulation $\bar{T}$ the "old" vertices $\mathbf{v}_i$ lie in and express them in barycentric coordinates with respect to the new tetrahedra constructed. Then linear interpolation is used to evaluate S_1 at an "old" vertex $\mathbf{v}_i$.

It is now quite simple to formulate an algorithm which iteratively removes data points.

ALGORITHM 1 : "DATA REDUCTION"

Input : Point set in three-dimensional space with associated function values and Delaunay triangulation, tolerance ε ;

Output : Reduced point set in three-dimensional space with associated function values and Delaunay triangulation ;

repeat until tolerance ε exceeded **or** no more tetrahedron with *true* predicate exists
(compute predicate $pred_t$ for all tetrahedra ;
for all tetrahedra with a *true* predicate **do**
((i) compute weight ω_t for tetrahedron t ;
(ii) determine tetrahedron $t_{\min}$ with minimal weight $\omega_{\min}$;
(iii) **if** $\omega_{\min} < \varepsilon$ **then**
compute triangulation $\bar{T}$ for point set $P_t \cup \{\mathbf{c}\}$
associated with tetrahedron $t_{\min}$;
)
)
compute the Delaunay triangulation for the reduced point set ;

It is possible to use the reduced data set and the resulting Delaunay triangulation after termination of algorithm 1 as input and start the reduction procedure again.

Some remarks have to be made concerning the data reduction algorithm. If two tetrahedra with a *true* predicate exist having both minimal weight $\omega_{\min}$ the result of the algorithm may depend on the decision which tetrahedron is chosen to be replaced by a single point. The data reduction algorithm given by Le Méhauté/Lafranche [29] requires a much more complicated strategy for retriangulating when extended to the trivariate case. When a point is removed the polyhedral boundary of its *platelet* (see

[29]) is not necessarily convex and to find the (locally) new triangulation is a quite involved process using their technique.

If the polyhedral boundary of the convex hull of the original point set does not need to be preserved, there are possibilities for removing tetrahedra having faces belonging to this polyhedral boundary, too. If a tetrahedron has one face on the polyhedral boundary, one could use the centroid of that face as the new point **c**, if it has two faces on the boundary, one could use the centroid of the edge shared by these two faces, if it has three faces on the boundary, one could use the vertex belonging only to this particular tetrahedron.

The effectiveness of the data reduction algorithm depends on the tolerance ε and on the " nature " of the data : if the data are originally obtained from a linear trivariate polynomial by some discretization process, the reduction will be greatest. In figure 2, the data reduction algorithm is shown for the bivariate case. The involved data and the old and new triangulation are represented.

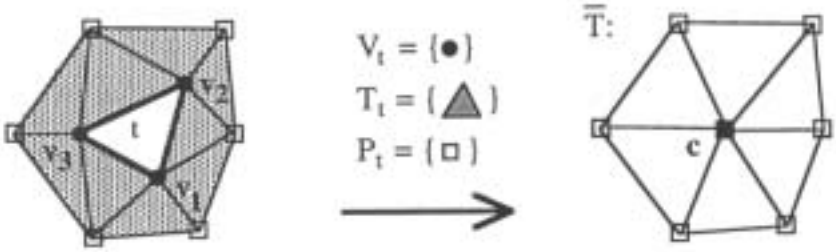

Figure 2. — Point removal in the bivariate case.

CONTOUR APPROXIMATION

Two different ways for obtaining a piecewise linear approximation to a contour $f = \text{const}$ must be considered for data of either scattered or rectilinear form. If scattered data are given or the original data set has been reduced as proposed in the previous chapter one is concerned with tetrahedra only. In the case of rectilinear data, the term " cube " will be used for a set of eight data points $\{\mathbf{x}_{i+I,j+J,k+K}\}$, $I, J, K \in \{0, 1\}$, associated with an arbitrary data point $\mathbf{x}_i$, even if the actual geometry of these eight points it not really a cube. Points lying on the approximation of the desired contour are now obtained by doing linear interpolation along edges of each tetrahedron (cube, respectively) : if one of the two points determining an edge has a function value greater and the other one smaller than the contour level, then linear interpolation is done along this edge to get a point on the contour approximation. It is assumed that the contour $f = \text{const}$ is *different from all given function values* at the vertices of all

tetrahedra (cubes). Special treatment is necessary if this is not the case. It will now be described how to obtain a piecewise planar approximation to the contour for both scattered and rectilinear data.

In the case of tetrahedra, one has to consider 2^4 cases (a function value is either greater or smaller than the contour and four vertices constitute a tetrahedron). By symmetry one actually has to treat two cases only : first, three function values are greater (smaller) and one function value is smaller (greater) than the contour level ; second, two function values are greater and two are smaller than the contour level. The first case automatically implies three points along edges of a tetrahedron which determine a contour triangle. The second case determines a planar quadrilateral that must be split into two triangles afterwards. These two cases for a tetrahedron are shown in figure 3 (black dots denote function values greater than the contour level, squares denote points on the contour).

Figure 3. — Contour triangles in a tetrahedron.

If rectilinear data are given and the data reduction procedure has not been applied, one is concerned with a total of 2^8 cases for each cube in the rectangular data grid. The rectangular data structure is essential for the contouring algorithm to be given now. It is possible to simply store all cases and then look them up in a table. Lorensen/Cline [33] use that technique and make use of symmetry among these 2^8 cases. Here, an algorithm will be given that automatically determines a piecewise planar approximation for the contour. Again, the first thing to be done is to determine contour points by applying linear interpolation between points of those cube edges whose associated function values imply the contour to intersect that edge (one value greater, the other one smaller than the contour level). Now, two definitions have to be given in order to understand the notation used in algorithm 2 that computes the polygonal boundaries (" contour polygons ") of a contour in a cube.

DEFINITION 1 : *Each contour point is considered lying on two cube faces of a particular cube. Two contour points belonging to the same cube have a* ***face in common*** *if among those four cube faces on which the two contour points lie (two cube faces per contour point) one cube face is the same.*

DEFINITION 2 : *Two contour points belonging to the same cube have a* ***corner in common*** *if the two contour points lie on cube edges that share a common vertex with an associated function value which is greater than the contour level.*

Definition 2 is necessary in order to construct consistent contour polygons. If there are more than two contour points on the same cube face it must be guaranteed that one connects always the same points forming contour polygons. The method for computing a triangular contour approximation presented by Lorensen/Cline [33] does not consider this consistency constraint. The meaning of the definitions of a common face and a common corner are illustrated in figure 4 : the pairs of points 1/2, 2/3, 3/4 and 1/4 each have a face in common, the points 5/6/7 have corner **c** in common.

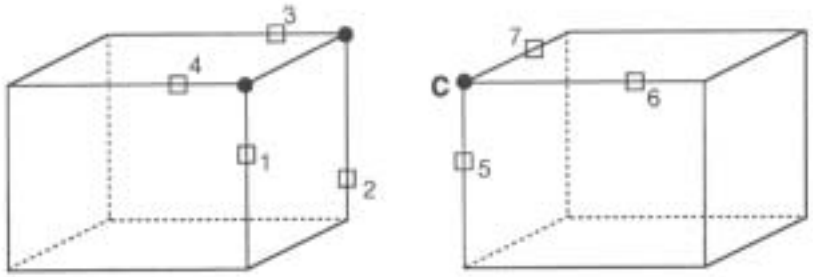

Figure 4. — Contour points with common face, contour points with common corner.

The idea of the algorithm is to construct a set of closed three-dimensional polygons that constitute the line boundaries for a piecewise planar approximation of the contour within a cube. The input for the described algorithm is a list of contour points on edges of a cube. It is further assumed that functions for effectively computing the two predicates " two points have a face in common " and " two points have a corner in common " are given. These functions are rather easy, they basically compare indices associated with two contour points referring to certain cube faces and corners. If the i-th point of the j-th polygon in a particular cube is denoted as p_i^j the algorithm proceeds as follows for each cube :

ALGORITHM 2 : " CONTOUR POLYGONS "

Input : List of contour points lying on edges of a particular cube

Output : Set of closed contour polygons with given contour points as vertices (the i-th point of the j-th polygon is denoted as p_i^j)

```
j := 1 ;
while not all contour points are associated with a polygon
  (  p_1^j := any contour point not yet associated with a polygon ;
     /* Among the two faces point p_1^j is lying on select one,      */
     /* denoted as f, determining the '' direction '' of the polygon. */
     determine face f ;
     p_2^j := the contour point having face f in common with p_1^j
          and not yet being assigned to another polygon
          if there is only one point on f not yet being assigned
               or
          the contour point having face f in common with p_1^j
          and having a corner in common with p_1^j
          and not yet being assigned to another polygon
          if there is more than one point on f not yet being assigned ;
     i := 3 ;
     while j-th polygon is not closed
       /* closed when all contour points on the next   */
       /* face f are assigned to some polygon      */
       (  /* if the face which p_{i-2}^j and p_{i-1}^j have in common is f_c */
          /* then the other face p_{i-1}^j is lying on is the next face f */
          determine next face f ;
          p_i^j := the contour point having face f in common with p_{i-1}^j
               and not yet being assigned to another polygon
               if there is only one point on f not yet being assigned
                    or
               the contour point having face f in common with p_{i-1}^j
               and having a corner in common with p_{i-1}^j
               and not yet being assigned to another polygon
               if there is more than one point on f not yet being assigned ;
          i := i + 1 ;
       )
     j := j + 1 ;
  )
```

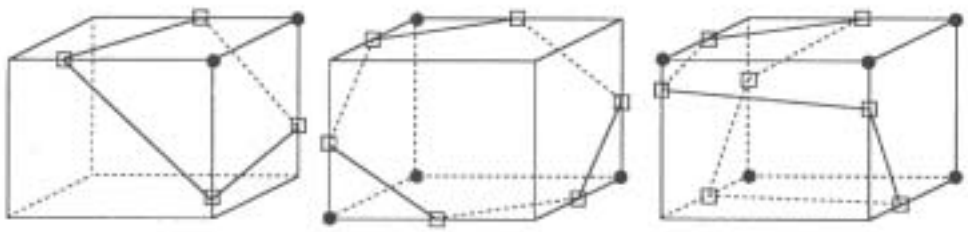

Figure 5. — Possible contour polygons inside cubes.

Algorithm 2 guarantees consistent contour polygons in the following sense : if the common face of two neighboring cubes contains two contour points (on two different edges of that face), there is only one way to connect these points ; if there are four contour points (one on each of the four edges of that face), the points will always be connected in the same fashion on that face since the convention for connecting contour points having corners in common (definition 2) is incorporated into algorithm 2. This algorithm might be slower than the method of Lorensen/Cline [33] based on a look-up table but it produces consistent contour polygons and might be extended easily for more complicated polyhedra than cubes. The technique of Lorensen/Cline [33] might lead to '' discontinuous '' triangulations (holes in the triangulation) as pointed out in Dürst [14] which are a result of constructing inconsistent contour polygons. Some possible contour polygons are shown in figure 5 (using algorithm 2).

At the end, a set of closed polygons is obtained (with three to seven vertices each) for a single cube. These polygons are interpreted as polygonal boundaries of a piecewise triangular approximation of the contour. Therefore, certain points of each polygon have to be connected to get the triangulation of the contour inside a cube. Because of consistency constraints with respect to neighboring cubes, a rule must be followed when computing the triangulation :

- The only edges connecting contour points on the same cube face in the triangulation of a contour polygon are the line segments constituting the contour polygon ; no other edges connecting contour points on the same face are allowed.

This rule guarantees that triangles consisting of three vertices on the same face are never constructed. This kind of triangles must be avoided in order to obtain a triangulation with a continuation into neighboring cubes. Possible triangulations inside single cubes for contour polygons are shown in figure 6.

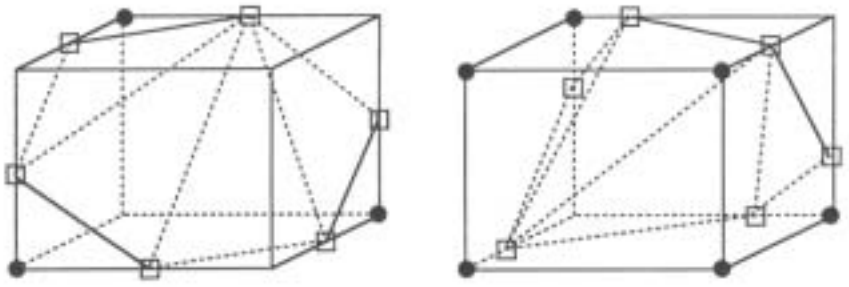

Figure 6. — Possible triangulations of contour polygons.

At this stage of the triangulation process of the contour, the '' quality '' of the triangulation within a single cube is not taken into account. As soon as

one has obtained the whole set of triangles approximating the contour, " smoothness " criteria can be used to improve the triangulation in the whole domain given by the convex hull of the data points. Referring to the involved data structures two tables have been created : one table contains all the contour points, the other table describes the triangulation by containing the references to those three contour points constituting each triangle. Figure 7 shows the piecewice linear approximation to the contour $f(x, y, z) = x^2 - y^2 + z^2 = 0.5$ obtained using algorithm 2 and the technique described for triangulating contour polygons. The trivariate function is evaluated on a $21 * 21 * 21$ grid, where $x, y, z \in [-1, 1]$. Flat shading is used for rendering.

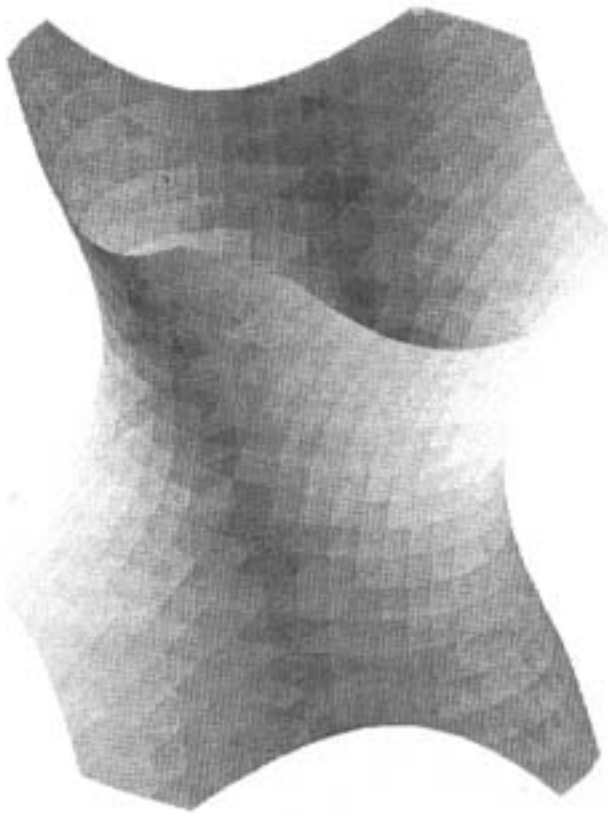

Figure 7. — Contour for $x^2 - y^2 + z^2 = 0.5$, $x, y, z \in [-1, 1]$.

The skull shown in figure 8 has been computed as a contour from a set of $68 * 64 * 64$ density measurements given in rectilinear form. The measurements $f_{i,j,k}$ themselves are integer values in $\{0, 1, ..., 255\}$. This contour (for $f = 13.5$) consists of approximately 30,000 contour points and 60,000 triangles. Gouraud shading is used for rendering.

Figure 8. — Human skull as contour of a density function.

TOPOLOGY

In this section it will be described how to obtain topological information for the set of (contour) triangles. Some surface schemes require neighborhood information in order to construct smooth interpolants. The topology is obtained as a '' byproduct '' of the contouring step. The contouring process basically yields two tables : a *vertex table* which contains each contour point as a triple (x, y, z) and a *triangle table* which contains each triangle as a triple (v_1, v_2, v_3) of indices referring to the three vertices in the vertex table constituting a triangle. An algorithm will be given that computes the neighborhood information considering the triangle table only. The contour $f = \text{const}$ might be split into several non-connected components inside the convex hull of the data points. Therefore, it is also essential to know to which component of the contour a particular triangle belongs to if one wants to model the different components separately.

DEFINITION 3 : *Two triangles t_i and t_j are* ***neighbors*** *if t_i have t_j and exactly two vertices in common.*

It is assumed that there are no degenerate cases meaning that a triangle has at most three neighbors (an edge in the triangulation is shared by at most two triangles). Otherwise, one would be dealing with a bifurcating triangulation. Algorithm 2 and the way each contour polygon inside a single cube has been triangulated guarantee that there are no bifurcations in this case. Denoting the total number of triangles by n, the algorithm to compute the number of neighbors for each triangle and the actual indices of these neighbors (referring to the triangle table) is straightforward :

ALGORITHM 3 : " NEIGHBORHOOD "

Input : Table of triangles, each given by its three vertex indices
Output : Number of neighbor triangles and their indices for each triangle in the triangle table

```
for i = 1 to n
  (  cnt := 0 ; /* number of neighbors */
     j := 1 ;
     while j ≤ n and cnt < 3
       (  if  i ≠ j and t_i and t_j are neighbors
              then
              (  cnt := cnt + 1 ;
                 cnt-th neighbor of t_i := j ;
              )
          j := j + 1 ;
       )
     number of neighbors for t_i := cnt ;
  )
```

Algorithm 3 is of order $O(n^2)$ with respect to the total number of triangles. Its performance can be improved by storing the index triple (i, j, k) for the " lower-front-bottom " vertex of the cube a triangle is lying in. Then, the search for the neighbors of a particular triangle inside a cube c can be restricted to the cube c itself and its six neighbor cubes in the rectilinear grid. By doing so order $O(n)$ is achieved. In order to give an algorithm for determining the component of the contour, a particular triangle belongs to a definition has to be given first.

DEFINITION 4 : *Two triangles t_i and t_j belong to the* ***same component of a contour*** *if triangles $t_1, t_2, \ldots, t_k$ exist, such that (t_1 is a neighbor of t_i) $\wedge$ (t_2 is a neighbor of t_1) $\wedge \ldots \wedge$ (t_k is a neighbor of t_{k-1}) $\wedge$ (t_j is a neighbor of t_k).*

The actual ordering of the triangles t_1, t_2, ..., t_k does not matter. In other words, two triangles t_i and t_j belong to the same component if there is a path from t_i to t_j of triangles which are pairwise neighbors to each other. To effectively determine the component index a particular triangle belongs to, the following algorithm is used. Again, n is the number of triangles.

ALGORITHM 4 : " COMPONENT OF CONTOUR "

```
Input  : Table of triangles (including the neighborhood information)
Output : Component index for each triangle which determines the compo-
         nent of the contour it belongs to
/* initially all triangles have an invalid component index " 0 " */
p := 1 ; /* first valid component index */
for i = 1 to n
  ( determine minimal component index min among all the component
    indices of t_i's neighbors ;
    if min = 0
      then
      ( component index for triangle t_i := p ;
        p := p + 1 ; /* one more component of contour found */
      )
      else
      ( component index for triangle t_i := min ;
        if there is 1 [are 2] valid component index [indices] ≠ min among
        t_i's neighbors
          then
          ( component index for this [these] triangle[s] := min ;
            p := p - 1 [2] ; /* " connecting " triangle has been found */
      )
  )
```

The case in brackets ([ ]) in algorithm 4 indicates the situation, when triangle t_i has three neighbors with valid component indices ($\neq 0$) which are all different from each other. At the end, each triangle will have been assigned to a certain component of the contour. The principle of assigning a component index to a triangle t_i is shown in figure 9 : the minimal component index among t_i's neighbors is 1 ; therefore, the component index for t_i is 1 and all component indices 2 and 3 are changed to component index 1.

At this stage of the modeling process, it might be worth considering some methods for improving the triangulation of the contour approximation. Knowing the neighborhood configuration the " max – min " or

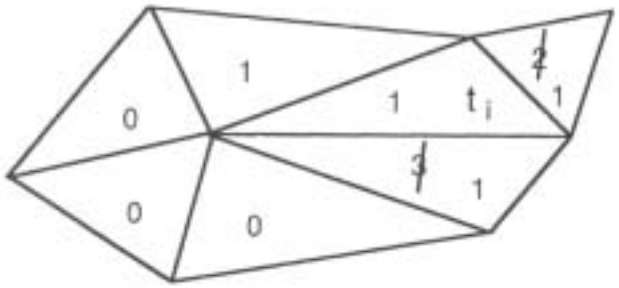

Figure 9. — Assigning component index to triangle t_i.

" min – max " angle criterion could be used. Iterative algorithms as proposed in Lawson [28] for swapping diagonals in quadrilaterals (given by two neighbor triangles) could be applied. Considering the fact that the triangulation to be improved is not a planar triangulation different optimization criteria might be appropriate. Choi *et al.* [11] use the angles between normals of neighboring triangles as a smoothness criterion and minimize these angles.

A G^1 SURFACE

The question to be addressed now is the problem of constructing a tangent plane continuous surface when a triangulation $\mathcal{T} = \{(v_1^t, v_2^t, v_3^t) | t = 1 \dots n_t\}$ (implying the topology and the neighborhood configuration) is given in form of index-triples (v_1^t, v_2^t, v_3^t) for a set of points $\mathbf{x}_i$ together with (outward) unit normal vectors $\mathbf{n}_i$, denoted as the set $XN = \{(\mathbf{x}_i, \mathbf{n}_i) | i = 1 \dots n\}$. The fact that each triangular patch has to be determined from three vertices and three normals solely gives rise to speak of a " six parameter patch ".

A brief overview for existing interpolation methods will be given. Different surface schemes have been proposed recently for solving the interpolation problem when points together with unit outward normals in a triangular mesh are given in three-dimensional space. It is not the goal here to discuss these methods in detail but rather to give a brief overview for existing methods. Basically, there are two approaches for solving the interpolation problem, either parametrically or implicitly defined triangular patches are created. The methods discussed in Farin [15], Hamann *et al.* [25], Nielson [35] and Piper [38] follow the first principle, Dahmen [12] uses the implicit form for constructing a surface based on the ideas of Sederberg [42]. If normal vectors are not initially given they can easily be approximated using a difference scheme or by differentiating a locally approximating function when the points to be interpolated are points on a contour of an underlying (unknown) trivariate function, as it is the case here. Possibilities

for obtaining gradient estimates for trivariate data are given in Stead [43] and Zucker/Hummel [46]. The gradients then determine the unit normal vectors for the contour points directly (the gradient is normal to a contour).

The methods of Hamann *et al.* [25] and Nielson [35] generate a tangent plane continuous surface for given points and unit outward normals. Each triangular patch is a convex combination of the form

$$\mathbf{s}(\mathbf{u}) = \sum_{i=1}^{3} w_i(\mathbf{u})\, \mathbf{s}_i(\mathbf{u}), \tag{7}$$

where the weight functions $w_i(\mathbf{u})$ and the single patch constituents $\mathbf{s}_i(\mathbf{u})$ are chosen in a way, such that each $\mathbf{s}_i(\mathbf{u})$, $i = 1 \ldots 3$, interpolates to all three boundary curves and to the normals along the i-th boundary curve of a patch. The notation "$\boldsymbol{u}$" refers to the triple (u_1, u_2, u_3) of barycentric coordinates associated with a point on the patch. The difference between the two methods is that Hamann *et al.* [25] use degree elevated conics in Bernstein-Bézier form and Nielson [35] uses cubic polynomials in Hermite form for the curve scheme that is needed in both methods for blending from a vertex to an opposite boundary curve of a triangular patch.

Piper [38] also constructs a tangent plane continuous surface when points and tangent (normal) vectors are given. His solution consists of a set of triangular Bernstein-Bézier patches of degree four. His scheme has to use a Clough-Tocher split to obtain the desired continuity.

Dahmen [12] uses the lowest degree possible for implicitly defined triangular Bernstein-Bézier patches with tangent plane continuity, which is degree two. These patches are contours of trivariate functions defined inside tetrahedra. To obtain these quadric patches a complicated procedure for constructing tetrahedra over each given triangle has to be followed. A Powell-Sabin split for each tetrahedron is necessary to guarantee tangent plane continuity between all quadric patches. The rendering process for these implicitly defined patches is quite expensive : intersections of paramet-

Figure 10. — Parametric triangular surfaces using different weights.

rically defined lines with the implicitly defined patch have to be calculated in order to obtain points on the surface.

An example for the method described in Hamann *et al.* [25] is shown in figure 10. Four points from a unit sphere have been given with their corresponding unit (outward) normal vectors. The weights for the interior Bézier points decrease in the sequence of surfaces from left to right giving the effect of increasing " tension ".

CURVATURE ESTIMATION

A technique will be described now that allows to interrogate a surface. It is assumed that the surface created in the previous step is approximated by a set of triangles (the topology is also known), and a method for curvature estimation is desired in order to interrogate the smoothness of the surface. Especially, when the number of approximating triangles is large and their size small it is hard to tell whether there are changes in the sign of Gaussian curvature by simply looking at an image of the set of triangles themselves. Therefore, a way is needed to approximate principal curvatures of a surface and to render these estimates on a screen.

Calladine [10] introduced a method for estimating Gaussian curvature for a triangulated surface. The technique described here will not be limited to this particular curvature measure but compute a whole set of estimates for normal curvatures for each surface point. Given a triangulation of a surface in three-dimensional space and outward (unit) normal vectors associated with each point in the triangulation, a technique will be derived that allows to calculate estimates for different kinds of curvature, e.g., Gaussian, mean and absolute curvature. These curvature estimates can then be linearly interpolated inside each triangle approximating the surface to be interrogated. Therefore, one obtains a Gouraud " shaded " representation for the curvature behavior of the surface. The curvature estimates will be color coded and rendered on a screen.

To obtain estimates for a particular point $\mathbf{x}_j$ only triangles having $\mathbf{x}_j$ as one among their three vertices have to be considered. It is assumned that the polygonal boundary around $\mathbf{x}_j$ formed by an *ordered sequence* of vertices (except $\mathbf{x}_j$ itself) constitutes a closed polygon, denoted as

$$\mathrm{pol}_j = \mathbf{x}_{p_1}\, \mathbf{x}_{p_2} \ldots \mathbf{x}_{p_j}\,, \tag{8}$$

where the edges of this polygon are given by $\overline{\mathbf{x}_{p_1}\, \mathbf{x}_{p2}}$, $\overline{\mathbf{x}_{p_2}\, \mathbf{x}_{p3}}$, ..., $\overline{\mathbf{x}_{p_j}\, \mathbf{x}_{p1}}$. The case when this polygon is *not closed* will be treated later. The normal vector at $\mathbf{x}_j$, denoted as $\mathbf{n}_j$, determines a whole set of vectors perpendicular to $\mathbf{n}_j$. If a finite set of perpendicular vectors, say

$$D_j = \left\{ \mathbf{d}_1^j, \mathbf{d}_2^j, \ldots, \mathbf{d}_k^j \right\}\,, \tag{9}$$

has to be determined, these vectors are chosen such that

(i) $\mathbf{n}_j \, \mathbf{d}_i^j = 0, \quad i = 1 \ldots k,$

(ii) $\mathbf{d}_1 = -\mathbf{d}_k,$

(iii) all angles between two consecutive vectors $\mathbf{d}_i^j$ and $\mathbf{d}_{i+1}^j$, $i = 1 \ldots k-1$, are the same, and

(iv) the sum of these angles equals 180 degrees.

The vector $\mathbf{d}_1$ can be chosen arbitrarily as long as it satisfies (i). The motivation for conditions (i) to (iv) is to obtain a finite number of normal planes with normal vectors $\mathbf{d}_i^j$ equidistantly distributed over a finite range (condition (iii)). The conditions (ii) and (iv) ensure that this range is just large enough to compute normal curvature estimates in all directions of interest and not to compute estimates in a certain direction more than once.

It is now very easy to get a normal curvature estimate for a point $\mathbf{x}_j$: calculate the two intersections between pol_j and a normal plane with normal vector $\mathbf{d}_i^j$, $i = 1 \ldots k$ (it is assumed that there are *exactly two intersections*). These two intersection points, called $\mathbf{y}_1$ and $\mathbf{y}_2$, determine a circle together with the point $\mathbf{x}_j$. Defining the two vectors $\mathbf{a} = \mathbf{x}_j - \mathbf{y}_1$ and $\mathbf{b} = \mathbf{y}_2 - \mathbf{y}_1$ such that $\mathbf{b} \times \mathbf{n}_j = \alpha \mathbf{d}_i^j$, $\alpha \geq 0$, the radius of this circle (approximating an osculating circle) is given by

$$r = \frac{\| \mathbf{a} \| \, \| \mathbf{b} \| \, \| \mathbf{a} - \mathbf{b} \|}{2 \| \mathbf{a} \times \mathbf{b} \|} \tag{10}$$

(see also Faux/Pratt [18]). Here, " $\| \ \|$ " is the usual euclidean norm. Therefore, the absolute value of the normal curvature estimate is given by $\kappa = 1/r$. The sign of the curvature estimate can be obtained doing the following convention : if $\mathbf{b} \times \mathbf{a} = \alpha \mathbf{d}_i^j$, $\alpha \geq 0$, κ is considered being positive, otherwise negative. Using this method for calculating normal curvature estimates, one gets a total of $k-1$ estimates for each point (the k-th estimate equals the 1st estimate). The minimal and maximal normal curvature estimates (κ_1 and κ_2) can be computed as estimates for the principal curvatures and estimates for Gaussian ($\kappa_1 \kappa_2$), mean $\left(\frac{1}{2} (\kappa_1 + \kappa_2) \right)$ and absolute curvature ($|\kappa_1| + |\kappa_2|$) can be derived from them. It has also been found that the average of all computed normal curvature estimates at a point $\mathbf{x}_j$, denoted as $\bar{\kappa}^j$, is a good measure to understand the curvature behavior of the surface (approximating the mean curvature) :

$$\bar{\kappa}^j = \frac{1}{k-1} \sum_{i=1}^{k-1} \kappa_i^j , \tag{11}$$

where κ_i^j, $i = 1 \ldots k-1$, are all normal curvature estimates computed for a point $\mathbf{x}_j$. Figure 11 illustrates the method for obtaining a single normal

curvature estimate for a point $\mathbf{x}_j$. The accuracy of the computed normal curvature estimates can further be checked using standard results of differential geometry, e.g., whether the normals for the normal planes associated with the principal curvature estimates κ_1 and κ_2 are perpendicular to each other, or whether Euler's theorem is satisfied (see Farin [17]). It will be investigated whether the method presented here can be extended to higher-dimensional geometry in order to interrogate higher-dimensional surfaces, e.g., surfaces of the form $(x, y, z, f(x, y, z))^T$ (see Rath [40]).

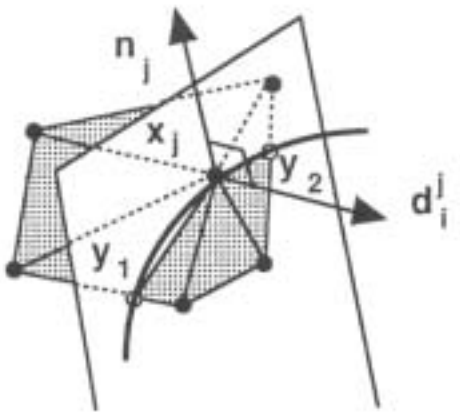

Figure 11. — Normal curvature estimate for x_j.

Some remarks have to be made. The quality of the curvature approximation depends on the size of the involved triangles relative to the surface area approximated by them and on the accuracy of the normal vectors at the data points. If $\mathbf{x}_j$ is *not surrounded by a closed polygon* two cases can occur : either, some among the used normal planes for obtaining normal curvature estimates still have two intersections or, no normal plane has two intersections with the polygon. In the first case, some normal curvature estimates can still be computed using the proposed technique, in the second case, a weighted average of curvature estimates of points connected to $\mathbf{x}_j$ by an edge in the triangulation could be taken. Special consideration must be given to the case when *more than two intersections* are found between a normal plane and a polygon surrounding $\mathbf{x}_j$. Figure 12 shows the average of all computed normal curvature estimates (formula (11)) for points on the surface $(x, y, f(x, y))^T$, where $f(x, y) = 0.2\ (x^2 - y^2)$ and $x, y \in [-1, 1]$. Normal vectors for a point on the surface can be computed exactly using the gradient of f and are given by $(-f_x(x, y), -f_y(x, y), 1)^T$. The function f is evaluated on a $21 * 21 * 21$ rectangular grid such that the surface is approximated by $20 * 20 * 2 = 800$ triangles (each rectangle in the domain is split into two triangles). Brighter grey levels represent areas with positive average normal curvature estimates, darker areas with negative average

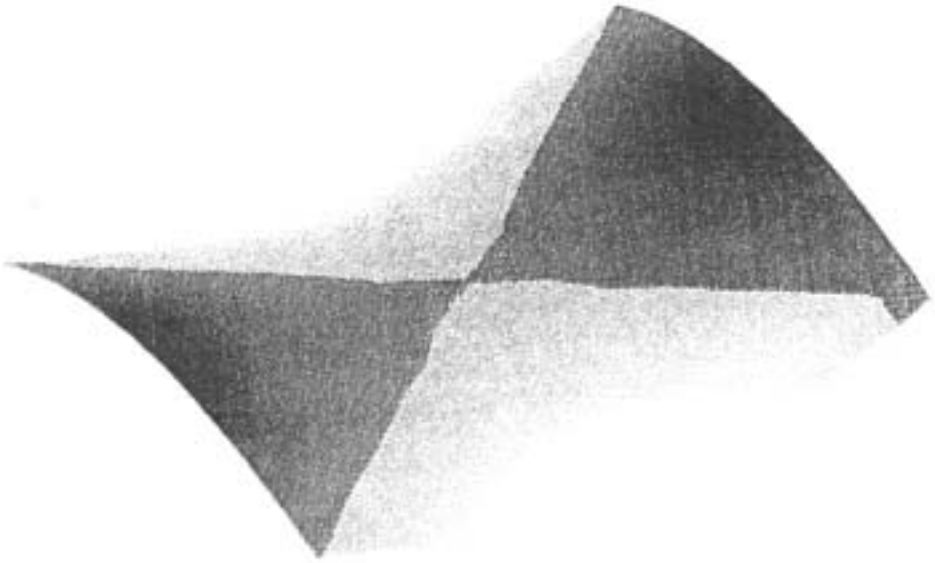

Figure 12. — Average normal curvature estimate for a bivariate function.

Figure 13. — Average normal curvature estimate on human skull (see fig. 8).

normal curvature estimates. The "discontinuity" in the grey levels corresponds to the change of sign in the estimate for the average normal curvature.

Very often, surfaces appear to have no changes in the sign of curvature behavior when rendered on a screen, especially when Gouraud shading is used. If there are changes in the sign of curvature these can be made visible using the described technique for surfaces approximated by triangular facets. In figure 13 the estimate for the average normal curvature (formula (11)) is shown for the human skull from figure 8 (which appears to be fairly smooth there). The triangulation used for computing the estimates is the triangulation obtained from the contour approximation step (ii). Normals for the points on the surface are obtained using a linear difference scheme for gradient approximation of the underlying trivariate function. Brighter grey levels represent areas with positive average normal curvature estimates, darker grey levels areas with negative estimates. The changes of sign in average normal curvature can be seen clearly (sudden changes in the grey levels).

CONCLUSIONS

A scheme has been presented which includes all steps necessary to mathematically model contours of skalar fields in three-dimensional space, beginning with data reduction and ending with a (contour) surface interrogation technique. The limited length for this publication and the wish not to leave out certain aspects of the whole modeling process have not allowed to discuss the single steps in more depth.

Future research is planned particularly for data reduction and curvature estimation. Papers treating the problems of computing a piecewise linear approximation for a contour and of curvature estimation are in preparation. The methods described more briefly here will be explained in greater detail here.

ACKNOWLEDGEMENTS

The work presented here was supported by the U.S. Department of Energy under contract DE-FG02-87ER25041 to Arizona State University and by NATO research grant RG 0097/88. Some of the algorithms described above have been implemented on a Silicon Graphics workstation 4D/220 GTX by the author (contour approximation, obtaining topological information, generating a G^1 surface, curvature estimation). The author also wants to thank all members of the computer-aided geometric design research group in the Computer Science Department at Arizona State University for their help.

REFERENCES

[1] P. ALFELD (1989), Scattered Data Interpolation in Three or More Variables, *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche, L. Schumaker (eds.), Academic Press, 1-33.

[2] A. A. BALL (1974), Consurf I, *Comput. Aided Design* 6, 243-249.

[3] A. A. BALL (1975), Consurf II, *Comput. Aided Design* 7, 237-242.

[4] A. A. BALL (1977), Consurf III, *Comput. Aided Design* 9, 9-12.

[5] R. E. BARNHILL, G. BIRKHOFF, W. J. GORDON (1973), Smooth Interpolation in Triangles, *J. Approx. Theory*, 8, 114-128.

[6] R. E. BARNHILL (1985), Surfaces in Computer Aided Geometric Design : A Survey with New Results, *Comput. Aided Geom. Design*, Vol. **2,** N° 1-3, 1-17.

[7] B. K. BLOOMQUIST (1990), Contouring Trivariate Surfaces, *Masters Thesis*, Arizona State University, Department of Computer Science.

[8] W. BOEHM, G. FARIN, J. KAHMANN (1984), A Survey of Curve and Surface Methods in CAGD, *Comput. Aided Geom. Design*, Vol. 1, 1-60.

[9] D. H. BALLARD, C. M. BROWN (1982), Computer Vision, Prentice Hall.

[10] C. R. CALLADINE (1986), Gaussian Curvature and Shell Structures, *Mathematics of Surfaces*, J. Gregory (ed.), Clarendon Press, Oxford, 179-196.

[11] B. K. CHOI, H. Y. SHIN, Y. I. YOON, J. W. LEE (1988), Triangulation of Scattered Data in 3D Space, *Comput. Aided Design*, Vol. 20, N° 5, 239-248.

[12] W. DAHMEN (1989), Smooth Piecewise Quadric Surfaces, *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche, L. L. Schumaker (eds.), Academic Press, 181-193.

[13] R. A. DREBIN, L. CARPENTER, P. HANRAHAN (1988), Volume Rendering, *Comput. Graphics*, Vol. 22, N° 4, 65-74.

[14] M. J. DÜRST (1988), Letters : Additional Reference to « Marching Cubes », *Comput. Graphics*, Vol. 22, N° 2.

[15] G. FARIN (1983), Smooth Interpolation to Scattered 3D Data, *Surfaces in CAGD*, R. E. Barnhill, W. Boehm (eds.), 43-63.

[16] G. FARIN (1986), Triangular Bernstein-Bézier Patches, *Comput. Aided Geom. Design*, Vol. 3, N° 2, 83-127.

[17] G. FARIN (1988), Curves and Surfaces for Computer Aided Geometric Design, Academic Press, Inc.

[18] I. FAUX, M. PRATT (1979), Computational Geometry for Design and Manufacture, Ellis Horwood.

[19] T. A. FOLEY, D. A. LANE, G. M. NIELSON (1990), Towards Animating Ray-Traced Volume Visualization, to appear in *Visualization and Computer Animation Journal*.

[20] R. FRANKE, G. M. NIELSON (1990), Scattered Data Interpolation and Applications : A Tutorial and Survey, *Geometric Modelling : Methods and Their Applications*, H. Hagen, D. Roller (eds.), Springer.

[21] K. S. FU, R. C. GONZALEZ, C. S. G. LEE (1987), Robotics, McGraw-Hill.

[22] H. FUCHS, M. LEVOY, S. M. PIZER (1989), Interactive Visualization of 3D Medical Data, *Computer*, Vol. 22, N° 8, 46-51.

[23] H. HAGEN, H. POTTMANN (1989), Curvature Continuous Triangular Interpolants, *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche, L. L. Schumaker (eds.), Academic Press, 373-384.

[24] B. HAMANN (1990), Visualisierungstechniken zur Darstellung dreidimensionaler Datenmengen (Visualization Techniques for the Representation of Three-Dimensional Data Sets, in German), in *CAD Computergraphik*, Vol. 14, N° 2, Austria, 129-139.

[25] B. HAMANN, G. FARIN, G. M. NIELSON (1990), A Parametric Triangular Patch Based on Generalized Conics, in : Farin, G., ed., NURBS for Curve and Surface Design, SIAM, Philadelphia, 75-85.

[26] J. HOSCHEK, D. LASSER (1989), Grundlagen der Geometrischen Datenverarbeitung (German), Teubner, Stuttgart (F.R.G.).

[27] J. KAJIYA, B. VON HERZEN (1984), Ray Tracing Volume Densities, *Comput. Graphics*, Vol. 18, N° 3, 165-173.

[28] C. L. LAWSON (1977), Software for C^1 Surface Interpolation, *Mathematical Software III*, J. R. Rice (ed.), Academic Press, 161-194.

[29] A. J. Y. LE MÉHAUTÉ, Y. LAFRANCHE (1989), A Knot Removal Strategy for Scattered Data in $\mathbb{R}^2$, *Mathematical Methods in Computer Aided Geometric Design*, T. Lyche, L. L. Schumaker (eds.), Academic Press, 419-426.

[30] M. LEVOY (1988), Display of Surfaces from Volume Data, *IEEE Comput. Graphics Appl.*, Vol. 8, N° 3, 29-37.

[31] M. LEVOY (1990), Volume Rendering, *IEEE Comput. Graphics Appl.*, Vol. 10, N° 2, 33-40.

[32] M. B. LONG, K. LYONS, J. K. LAM (1989), Acquisition and Representation of 2D and 3D Data from Turbulent Flows and Flames, *Computer*, Vol. 22, N° 8, 39-45.

[33] W. E. LORENSEN, H. E. CLINE (1987), Marching Cubes : A High Resolution 3D Surface Construction Algorithm, *Comput. Graphics*, Vol. 21, N° 4, 163-169.

[34] D. R. NEY, E. K. FISHMAN, D. MAGID, R. A. DREBIN (1990), Volumetric Rendering, *IEEE Comput. Graphics Appl.*, Vol. 10, N° 2, 24-32.

[35] G. M. NIELSON (1987), A Transfinite, Visually Continuous, Triangular Interpolant, *Geometric Modeling : Algorithms and New Trends*, G. Farin (ed.), SIAM Publications, Philadelphia, 235-246.

[36] G. M. NIELSON, B. HAMANN (1990), Techniques for the Interactive Visualization of Volumetric Data, *Proceedings of the IEEE Conference Visualization '90*, IEEE Computer Society Press, 45-50.

[37] C. S. PETERSEN, B. R. PIPER, A. J. WORSEY (1987), Adaptive Contouring of a Trivariate Interpolant, *Geometric Modeling : Algorithms and New Trends*, G. Farin (ed.), SIAM Publications, Philadelphia, 385-395.

[38] B. R. PIPER (1987), Visually Smooth Interpolation with Triangular Bézier Patches, *Geometric Modeling : Algorithms and New Trends*, G. Farin (ed.), SIAM Publications, Philadelphia, 221-233.

[39] H. POTTMANN (1989), Scattered Data Interpolation Based upon Generalized Minimum Norm Networks, submitted to *Constructive Approximation Theory*.

[40] W. RATH (1988), Computergestützte Darstellungen von Hyperflächen des $\mathbb{R}^4$ und deren Anwendungsmöglichkeiten im CAGD (German), *CAD Computergraphik*, Vol. 11, N° 4, 111-117.

[41] P. SABELLA (1988), A Rendering Algorithm for Visualizing 3D Scalar Fields, *Comput. Graphics*, Vol. 22, N° 4, 51-55.

[42] T. W. SEDERBERG (1985), Piecewise Algebraic Surface Patches, *Comput. Aided Geom. Design*, Vol. 2, N° 1-3, 53-59.

[43] S. E. STEAD (1984), Estimation of Gradients from Scattered Data, *Rocky Mountain J. Math.*, Vol. 14, N° 1, 265-279.

[44] U. TIEDE, K. H. HÖHNE, M. BOMANS, A. POMMERT, M. RIEMER, G. WIEBECKE (1990), Surface Rendering, *IEEE Comput. Graphics Appl.*, Vol. 10, N° 2, 41-53.

[45] A. WORSEY, G. FARIN (1987), An N-Dimensional Clough-Tocher Element, *Constr. Approx.*, Vol. 3, 99-110.

[46] S. W. ZUCKER, R. A. HUMMEL (1981), A Three-Dimensional Edge Operator, *IEEE Trans. Pattern Recogn. Mach. Intelligence*, Vol. PAMI-3, N° 3, 324-331.