RECONSTRUCTION OF SURFACES AND SURFACES-ON-SURFACES FROM UNORGANIZED WEIGHTED POINTS

Chandrajit L. Bajaj
Fausto Bernardini
Guoliang Xu

CSD-TR-94-001
January 1994

Reconstruction of Surfaces and Surfaces-on-Surfaces from Unorganized Weighted Points*

Chandrajit L. Bajaj Fausto Bernardini Guoliang Xu

Department of Computer Sciences
Purdue University
West Lafayette, Indiana

bajaj@cs.purdue.edu, Tel: 317-494-6531

Abstract

We present a unified approach for the reconstruction of both a C^1 smooth domain surface S_d and a C^1 smooth function S_f (surface-on-surface) on the domain surface S_d from an unorganized collection of weighted points $\{(x_i, y_i, z_i, w_i)\}$. The set of points $P = \{(x_i, y_i, z_i)\}$ are assumed to be sampled from on or near an unknown domain S in $\mathbb{R}^3$ while the weights w_i are assumed sampled from some unknown scalar function F on the domain S. Examples include the pressure F on the wing S of an airplane or the temperature T on a portion S of the human body. The simplicity and unified nature of our algorithm arises from several uses of appropriate sub-structures of the three-dimensional Delaunay Triangulation, and its dual the three-dimensional Voronoi diagram. The C^1 smooth surface S_d and the C^1 smooth function S_f are constructed using trivariate cubic Bezier patches. We also present techniques for efficiently producing different visualizations of the reconstructed surface S_d and the surface-on-surface S_f.

*This work was supported in part by NSF grants CCR 92-22467, DMS 91-01424, AFOSR grants F49620-93-10138, F49620-94-1-0080, NASA grant NAG-1-1473 and a gift from AT&T.

1 Introduction

We consider the following problem:
Given an unorganized collection of weighted points $\{(x_i, y_i, z_i, w_i)\} \subset \mathbb{R}^4$, $i = 1 \ldots n$, *where the set of points* $P = \{(x_i, y_i, z_i)\} \subset \mathbb{R}^3$ *are assumed to be sampled from on or near an unknown domain* S *in* $\mathbb{R}^3$ *while the weights* $\{w_i\} \subset \mathbb{R}^1$ *are assumed sampled from some known or computed scalar function* F *on the domain* S, *construct a* C^1 *smooth surface* $S_d : F^{(1)}(x, y, z) = 0$ *and a* C^1 *smooth function (surface-on-surface)* $S_f : F^{(2)}(x, y, z)$ *on some domain that contains* P *such that*

1. $F^{(1)}(x_i, y_i, z_i) = 0$
2. $F^{(2)}(x_i, y_i, z_i) = w_i$

Additionally, generate different visualizations of the domain surface S_d *and the surface-on-surface* S_f.

Reconstructing the domain surface S_d from unorganized points in $\mathbb{R}^3$ is a fundamental problem in CAD and computer vision [10, 25, 33]. These papers provide a very nice survey of both the varied nature of applications and past approaches. The new techniques introduced in [10, 25, 33] are for piecewise-linear C^0 reconstructions of the unknown surface S. In this paper we construct a C^1 smooth domain surface S_d using piecewise cubic, triangular implicit Bezier patches (the zero contour of a C^1 trivariate piecewise Bezier function). Related prior work [2, 4, 5, 12, 13, 23, 24] of fitting with smooth implicit surface patches, minimally all require an input surface triangulation of the data points. The paper of [26] is similars to ours in that it only assumes a sufficiently dense set of input data points but differs from our approach in the adaptive nature of refinement, in time efficiency and in the degree of the implicit surface patches used . Paper [26] uses an octree like static subdivision scheme and then uses tri-quadratic (degree six) tensor product surface patches to achieve C^1 continuity. Our scheme effectively utilizes the incremental Delaunay triangulation for a more adaptive fit; the dual Voronoi diangram for efficient point location in signed distance computations and degree three implicit surface patches. Furthermore, in the same time it computes a C^1 smooth approximation S_f of the sampled scalar function.

If the surface S_d is given, the problem of constructing the scalar function S_f is known as surface interpolation on a surface, and arises in several application areas. For example, in modeling and visualizing the rain fall on the earth, the pressure on the wing of an airplane or the temperature on a human body. Note that the trivariate scalar function S_f is a two dimensional surface in $\mathbb{R}^4$ since its domain is the two dimensional surface S_d (and not all of $\mathbb{R}^3$). The problem is relatively recent and was posed as an open question by Barnhill [6]. A number of methods have been developed since then for dealing with the problem (for surveys see [7, 22, 28]). Most of the solutions interpolate scattered data over planar or spherical domain surfaces. In [9] and [21], the domain surface is generalized to a convex surface and a topological genus zero surface, respectively. Pottmann [30] presents a method which does not possess similar restrictions on the domain surface but requires it to be at least C^2 differentiable. In [8] the C^2 restriction is dropped, however the interpolation surface is constructed by transfinite interpolation using non-polynomials. A similar non-polynomial transfinite interpolant construction is used in [27] while [32] requires at least C^4 for his interpolation. A nice survey of existing trivariate interpolation methods is given in [28].

Our domain surface S_d and surface-on-surface S_f reconstruction scheme does not impose any convexity or differentiability restrictions on the original domain surface S or function F, except that it assumes that there is a sufficient sampling of the input point data to unambiguously reconstruct the domain surface S_d. While it is difficult to precisely bound the required sampling density, we

address this issue in section 4.3 and characterize the required sampling density in terms of an α-shape (subgraph of a Delaunay triangulation of the points) which matches the topology of the original (unknown) sampled surface S. Compared to the above methods our algorithm thus has the following advantages:

1. it unifies the reconstruction of the domain surface S and the scalar function F defined on the domain surface
2. It is capable of handling an unorganized collection of data points
3. It is adaptive and approximates large dense data sets with a relatively small number of C^1 smooth patches

The rest of our paper is as follows. In Section 2 we introduce some notation, facts and lemmas that are pre-requistes to the reconstruction algorithm. In Section 3, we present an outline of the reconstruction algorithm, and detail the main steps in section 4 upto the level of an implementation. Finally, in section 5 we describe techniques for visualizing the reconstructed surfaces and surface-on-surfaces and present examples of our implementation.

2 Algorithm Pre-requisites

We briefly recall definitions and properties needed for the reconstruction algorithms. The style of this presentation is informal. The reader can refer to the cited papers for a more detailed explanation of these concepts.

Delaunay Triangulations: Given a set P of points in R^3 one can build a tetrahedralization of the convex hull of P, that is, a partition of $conv(P)$ into tetrahedra, in such a way that the circumscribing sphere of each tetrahedron T does not contain any other point of P than the vertices of T. Such a tetrahedralization is called a (3D) Delaunay triangulation and, under non degeneracy assumptions (no three points on a line, etc.) it is unique. Many different techniques have been proposed for the computation of Delaunay triangulations (see [15, 31]). These techniques are mainly used and analyzed in the plane, but many of them are easily generalized to work in higher dimensions. For our purposes, an incremental approach is particularly well-suited, as it can be used in both a preprocessing step and the incremental refining of the adaptive, approximating triangulation (see section 4.1).

The algorithm we use is the randomized, incremental, flipping-based algorithm proposed in [18]. This algorithm uses a data structure, called the history DAG, that maintains the collection of discarded tetrahedra. At the beginning the triangulation is initialized as a single tetrahedron, with vertices "at infinity", that contains all points of P. At each step, the DAG is used to locate the tetrahedron that the point to be inserted lies in. Then the tetrahedron is split and the Delaunay property is re-established by flipping tetrahedra. The newly created tetrahedra are added as leaves to the DAG.

One can build the Delaunay triangulation of a set of n points in R^3 in $O(nlogn + n^2)$ expected time. The second term in this expression is of the same order as the maximum number of possible simplices. The expectation is taken over all possible sequences of the same n points, and is therefore independent of the points distribution. The actual expected time, that depends on the particular distribution the input sequence, can be much less than this.

Voronoi Diagrams: Voronoi diagrams are well known tools in computational geometry (see [3] for a survey). They provide an efficient solution to the *Post Office Problem*, that is an answer to the query: what is the closest point $p \in P$ to a given point q? Voronoi diagrams are related to Delaunay triangulations by duality. It is easy to build a Voronoi diagram once one has the corresponding triangulation, and vice-versa. A Voronoi diagram is a partition of the space in convex cells. There is a cell for each point of $p \in P$, and the cell of a point p is the set of points that are closer to p than to any other point of P. So, all one has to do to answer the closest-point query is to locate the cell the query point lies in. Efficient point-location data structures can be built on top of the Voronoi diagram with no significant effort. Using the randomized approach described in [11], one builds the point-location data-structure (called an *RPO-tree*, for Randomized Post Office tree) on top of the Voronoi diagram in $O(n^{2(1+\epsilon)})$ expected time, for any fixed $\epsilon > 0$, and is then able to answer the closest-point query in $O(\log n)$ expected time. The data structure requires $O(n^{2(1+\epsilon)})$ space in the worst case. We use the *RPO-tree* data structure for our point location and signed distance computations (see section 4.2).

α-Shapes: Given the Delaunay triangulation T of a point set P, regarded as a simplicial complex, one can assign to each simplex $\sigma \in T$ (vertices, edges, triangles and tetrahedra) a *size* defined as the square of the radius of the smallest sphere containing σ.

The subcomplex Σ_α of simplices σ with one of the following properties: (a) the size of σ is less than α or (b) σ is a face of τ and $\tau \in \Sigma_\alpha$, is called the α-shape of P. α-Shapes have been introduced in the plane by [17] and then extended to any dimensions and to weighted sets of points (see [16]).

One can intuitively think of an α-shape as the subcomplex of T obtained in the following way: imagine that a ball-shaped eraser, whose radius is $\sqrt{\alpha}$, is moved in the space, assuming all possible positions such that no point of P lies in the eraser. The eraser removes all simplices it can pass through, but not those whose size is smaller than α. The remaining simplices (together with all their faces) form the α-shape for that value of the parameter α. Notice that there exists only a finite number of different α-shapes. The collection of all possible α-shapes of P is called the *family* of α-shapes of P. We use the α-shape computation for our generating an inital piecewise linear approximation of the domain surface S_d (see section 4.3).

The following facts and lemmas are used in section 4.4 for constructing the C^1 smooth domain surface S_d and surface-on-surface S_f.

Bernstein–Bezier (BB) Form: Let p_1, p_2, p_3, $p_4 \in I\!R^3$ be affine independent. Then the tetrahedron with vertices p_1, p_2, p_3, and p_4, is $V = [p_1 p_2 p_3 p_4]$. For any $p = \sum_{i=1}^{4} \alpha_i p_i \in V$, $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$ is the barycentric coordinate of p. Let $p = (x, y, z)^T$, $p_i = (x_i, y_i, z_i)^T$. Then the barycentric coordinates relate to the Cartesian coordinates via the following relation

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} \tag{2.1}$$

Any polynomial $f(p)$ of degree n can be expressed as Bernstein-Bezier(BB) form over V as

$$f(p) = \sum_{|\lambda|=n} b_\lambda \; B_\lambda^n(\alpha), \quad \lambda \in \mathcal{Z}_+^4 \tag{2.2}$$

where

$$B_\lambda^n(\alpha) = \frac{n!}{\lambda_1!\lambda_2!\lambda_3!\lambda_4!} \; \alpha_1^{\lambda_1}\alpha_2^{\lambda_2}\alpha_3^{\lambda_3}\alpha_4^{\lambda_4} \tag{2.3}$$

is Bernstein polynomial, $|\lambda| = \sum_{i=1}^4 \lambda_i$ with $\lambda = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)^T$, $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$ is barycentric coordinate of p, $b_\lambda = b_{\lambda_1\lambda_2\lambda_3\lambda_4}$(as a subscript, we simply write λ as $\lambda_1\lambda_2\lambda_3\lambda_4$) are called control points, and $\mathcal{Z}_+^4$ stands for the set of all four dimensional vectors with nonnegative integer components. The following basic facts about the BB form will be used in this paper.

Lemma 2.1 ([23]). *If* $f(p) = \sum_{|\lambda|=n} \; b_\lambda B_\lambda^n(\alpha)$, *then*

$$b_{(n-1)e_i+e_j} = b_{ne_i} + \frac{1}{n}(p_j - p_i)^T \nabla f(p_i), \quad j = 1,2,3,4; \quad j \neq i \tag{2.4}$$

This lemma is used to determine the Bezier coefficients around the vertex from the gradient at the vertex.

Lemma 2.2 ([20]). *Let* $f(p) = \sum_{|\lambda|=n} \; a_\lambda B_\lambda^n(\alpha)$ *and* $g(p) = \sum_{|\lambda|=n} \; b_\lambda B_\lambda^n(\alpha)$ *be two polynomials defined on two tetrahedra* $[p_1p_2p_3p_4]$ *and* $[p_1'p_2p_3p_4]$, *respectively. Then*
(i) f *and* g *are* C^0 *continuous at the common face* $[p_2p_3p_4]$ *if and only if*

$$a_\lambda = b_\lambda, \quad for \quad any \quad \lambda = 0\lambda_2\lambda_3\lambda_4, \quad |\lambda| = n \tag{2.5}$$

(ii) f *and* g *are* C^1 *continuous at the common face* $[p_2p_3p_4]$ *if and only if (2.5) holds and*

$$b_{1\lambda_2\lambda_3\lambda_4} = \beta_1 a_{1\lambda_2\lambda_3\lambda_4} + \beta_2 a_{0\lambda_2\lambda_3\lambda_4+0100} + \beta_3 a_{0\lambda_2\lambda_3\lambda_4+0010} + \beta_4 a_{0\lambda_2\lambda_3\lambda_4+0001} \tag{2.6}$$

where $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)^T$ *are defined by the following relation*

$$p_1' = \beta_1 p_1 + \beta_2 p_2 + \beta_3 p_3 + \beta_4 p_4, \quad |\beta| = 1$$

The relation (2.6) will be called coplanar condition.

Lemma 2.3. *Let* $f(p) = F(\alpha) = \sum_{|\lambda|=n} \; b_\lambda B_\lambda^n(\alpha)$ *with* α *is the barycentric coordinates of* p. *Then for any given* $p^{(1)}$ *and* $p^{(2)}$, *let* $\alpha^{(1)}$ *and* $\alpha^{(2)}$ *be their barycentric coordinates. Then*

$$\begin{aligned} \nabla f(p)^T(p^{(1)} - p^{(2)}) &= \nabla F(\alpha)^T(\alpha^{(1)} - \alpha^{(2)}) \\ &= n \sum_{|\lambda|=n-1} b_\lambda^1(\alpha^{(1)} - \alpha^{(2)}) B_\lambda^{n-1}(\alpha) \end{aligned}$$

where

$$b_\lambda^1(\alpha^{(1)} - \alpha^{(2)}) = \sum_{|j|=1} b_{\lambda+j} B_j^1(\alpha^{(1)} - \alpha^{(2)})$$

The first equality of the lemma can easily be proved by the transform (2.1). The second equality can be found in [19]. The lemma is used to ensure that the polynomial $F(\alpha)$ has the specified directional derivative $\nabla f(p)^T(p^{(1)} - p^{(2)})$ at a given point.

3 Outline of the Algorithm

The algorithm constructs an incremental Delaunay triangulation $\mathcal{T}$ over which piecewise C^1 continuous functions $F^{(1)}$ and $F^{(2)}$ are generated, respectively. The steps described below and in the next section are with sufficient detail for an implementation.

Algorithm 1 *1. Build an initial bounding tetrahedron $\mathcal{T} = \{T\}$, such that $P \subset T$. Set $V =$ vertices of T.*

2. For each $T \in \mathcal{T}$, if $T \cap P \neq \emptyset$ and T does not have an interpolant $f_T^{(2)}$, build a local interpolant $f_T^{(2)}(p) = \sum_{i+j+k+l=m} b_{ijkl}^{(2)} B_{ijkl}^m(\alpha)$, such that

$$f_T^{(2)}(p_i) = w_i, \quad p_i \in P \cap T$$

in the least square sense, where $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)^T$, $p = \sum_{i=1}^4 \alpha_i p_i$. The degree m of $f_T^{(2)}$ is chosen to be linear, quadratic or cubic depending on the the number of points in $P \cap T$ and the error $\epsilon_T^{(2)}$ given by

$$\epsilon_T^{(2)} = \frac{\sqrt{\sum_{p_i \in P \cap T} (f_T^{(2)}(p_i) - w_i)^2}}{Card(P \cap T)}$$

If $T \cap P = \emptyset$, set $\epsilon_T^{(2)} = 0$.

3. Find a tetrahedron $T \in \mathcal{T}$, such that

$$\epsilon_T^{(2)} = \max_{T' \in \mathcal{T}} \{\epsilon_{T'}^{(2)}\}$$

If $\epsilon_T^{(2)}$ is within the given error limit ϵ, that is $\epsilon_T^{(2)} \leq \epsilon$. Then do step 4. Otherwise, find the center point p_T of the circumscribing sphere of T. Then add p_T to the vertex set V and update the Delaunay triangulation. (Adding the center of the circumscribing sphere of T is utilizing the empty sphere property of Delaunay triangulations and in general yields good aspect ratio tetrahedra in the final triangulation [14]) Then go back to step 2.

4. For each $T \in \mathcal{T}$, do the following:

a. if T does not have a local interpolant $f_T^{(1)}$ on it, then compute the signed distances d_{ijkl} (see section 4.2 for details) for the regular points p_{ijkl} of T, that is

$$p_{ijkl} = \frac{i}{n} p_1 + \frac{j}{n} p_2 + \frac{k}{n} p_3 + \frac{l}{n} p_4, \; i + j + k + l = n$$

where n is the degree of the polynomial to be used, $p_i \in V$ is the vertex of T. Again, n is chosen to be linear, quadratic or cubic depending on the the number of points in $P \cap T$ and the error $\epsilon_T^{(1)}$. Define the degree n polynomial interpolant $f_T^{(1)}(p) = \sum_{i+j+k+l=n} b_{ijkl}^{(1)} B_{ijkl}^n(\alpha)$ by

$$f_T^{(1)}(p_{ijkl}) = d_{ijkl}, \; i + j + k + l = n$$

and the error $\epsilon_T^{(1)}$ by

$$\epsilon_T^{(1)} = \frac{\sqrt{\sum_{p \in P \cap T} f_T^{(1)}(p)^2}}{Card(P \cap T)}$$

b. if $T \cap P \neq \emptyset$ and T does not have an interpolant $f_T^{(2)}$ on it, build an interpolant $f_T^{(2)}$ and compute the error $\epsilon_T^{(2)}$ as in step 2. If $T \cap P = \emptyset$, set $\epsilon_T^{(2)} = 0$.

5. *Find a tetrahedron $T \in \mathcal{T}$, such that*

$$\epsilon_T = \max_{T' \in T} \{\epsilon_{T'}^{(1)} + \epsilon_{T'}^{(2)}\}$$

If ϵ_T is within the given error limit, that is $\epsilon_T \leq 2\ \epsilon$. Then go to step 6. Otherwise, find the center point p_T of the circumscribing sphere of T. Add p_T to the vertex set V, update the Delaunay triangulation and go back to step 4.

6. *Produce C^1 data $D^{(2)}$. That is for each vertex, if its neighbor tetrahedra have local interpolants $f_T^{(2)}$, compute function value $F^{(2)}$ and gradient $\nabla F^{(2)}$ by evaluating the interpolants $f_T^{(2)}$ at the vertex, and averaging the values. The averaging may be weighted:*

   a. *the weights can depend on the number of points in T*

   b. *can also depend on the error of the approximation of the interpolants*

   For the middle point of each edge, if its neighbor tetrahedra have interpolants $f_T^{(2)}$, compute each of the gradients $\nabla f_T^{(2)}$ there and then do averaging as before.

7. *Produce C^1 cubic interpolants $F_T^{(2)}$ from C^1 data $D^{(2)}$ for $T \in \mathcal{T}$ using the trivariate Clough-Tocher scheme (see section 4.4 for details). Then $F^{(2)}$ is defined by*

$$F^{(2)}\Big|_T = F_T^{(2)}$$

   If the C^1 data for T is not complete, that is if some vertices or edge middle points do not have data, use zero data instead.

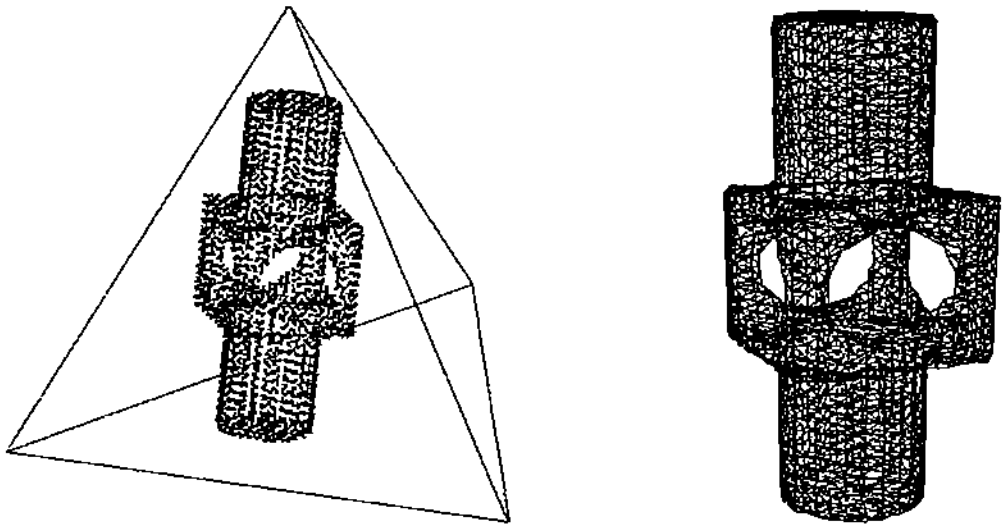
8. *Produce C^1 data $D^{(1)}$. That is for each vertex if its neighbor tetrahedra have local interpolants $f_T^{(1)}$, compute function value $F^{(1)}$ and gradient $\nabla F^{(1)}$ and gradient at the middle point of each edge from $f_T^{(1)}$, and similar to step 6.*

9. *Produce C^1 cubic interpolants $F_T^{(1)}$ for each T, from the C^1 data $D^{(1)}$ using the trivariate Clough-Tocher scheme (see section 4.4 for details), and similar to step 7. Then $F^{(1)}$ is defined by*

$$F^{(1)}\Big|_T = F_T^{(1)}$$

10. *For each $T \in \mathcal{T}$, display the surface $F_T^{(1)} = 0$, surface-on-surface $F_T^{(2)}$ over the first surface and draw iso-curves $F_T^{(2)} = w$ on the first surface. (See section 5 for details).*

Note: The incremental Delaunay triangulation in step 3 and step 5 also remembers the fact that several tetrahedron are unchanged during a single point insertion and re-triangulation. This way one avoids re-computation of the signed distances and the interpolants for the unchanged tetrahedra.



(a) (b)

Figure 4.1: Sampled Points of a Mechanical Part and its Piecewise Linear Approximation

4 Details of the Algorithm

4.1 Incremental Adaptive Refinement

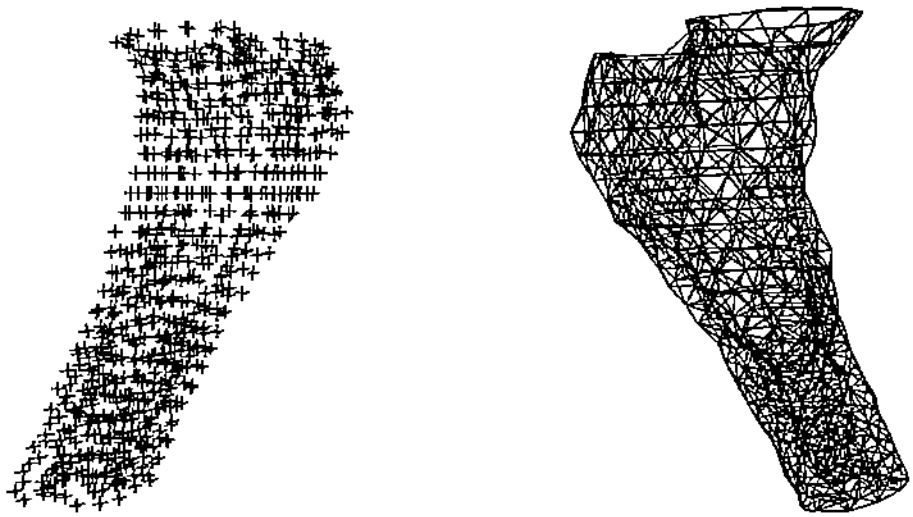
Steps 3 and 5 of the algorithm require the incremental refinement of the adaptive, approximating tetrahedral mesh. This is done by adding at each step a new point to split the tetrahedron with the maximum error, and using the incremental Delaunay triangulation algorithm to update the triangulation. One additional task is required after each splitting or flipping of tetrahedra takes place: the subset $P \cap \tau$ of points that lie within each modified simplex τ has to be updated. This is done by considering the points originally within the modified simplex, and reclassifying them with respect to the splitting/flipping planes.

4.2 Computation of the Signed Distances

Step 4 of the algorithm requires the computation of the signed distance of a query point (one of the Bezier domain points p_{ijkl} of a given tetrahedron treated as a trivariate triangular Bezier patch) with respect to the data set P. The signed distance problem can be formally stated as follows:

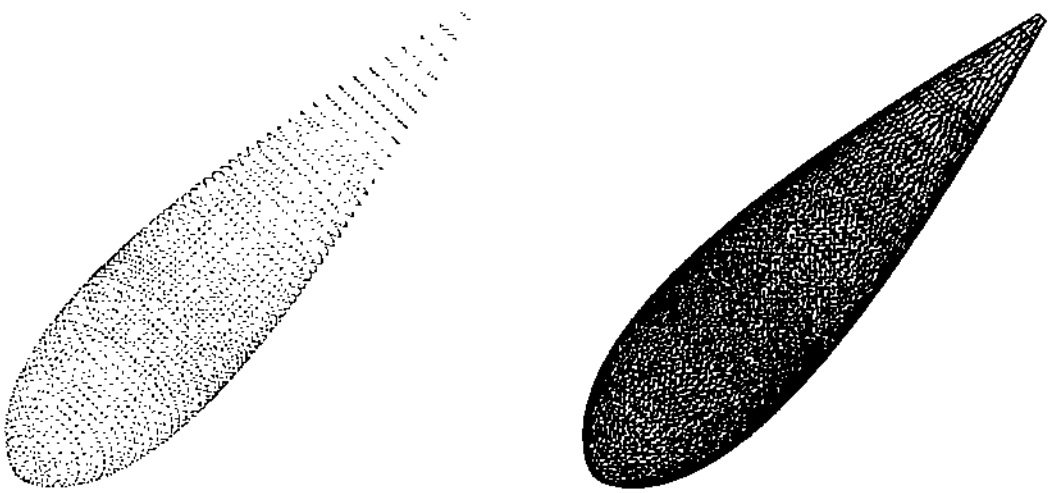
> Let a data set $P = \{(x_i, y_i, z_i)\}$ and a domain surface S_d that interpolates P be given. We assume that S_d divides the space into two disconnected regions, so that one can arbitrarily assign a distinct sign, say $+$ for internal and $-$ for external, to these regions. For a query point q, return the signed distance d_q such that $|d_q| = min_{p \in P}|qp|$ where $|qp|$ is the Euclidean distance between q and p, and the sign of d_q is chosen accordingly to the sign of the space region that q lies in.

Our approach to the solution of this problem is the use of the Delaunay triangulation for the Bezier tetrahedra, the Voronoi diagram to do fast point location, and the use of α-shapes as a means to build a piecewise linear approximation to the surface S_d to be used in signed distance



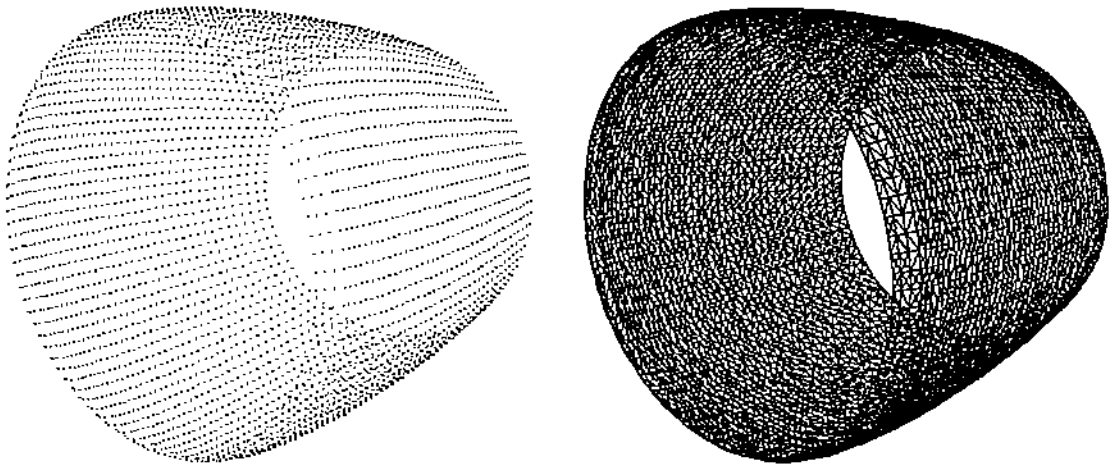
(a) (b)

Figure 4.2: Sampled Points of a Human Femur and its Piecewise Linear Approximation



(a) (b)

Figure 4.3: Sampled Points of a Jet Engine Nose and its Piecewise Linear Approximation



(a) (b)

Figure 4.4: Sampled Points of a Jet Engine Outer Cowl and its Piecewise Linear Approximation

queries. Note that all three structures are intimately related. Since the computation of the signed distance can be requested a considerable number of times, it pays to perform a preprocessing step to speed up the point location queries.

In step 0 of the algorithm the data set P is preprocessed, and the history DAG and the RPO-tree data structures are built to represent its Delaunay triangulation T, Voronoi diagram and family of α-shapes. Tetrahedra in the Delaunay triangulation are classified as either internal or external (and assigned a corresponding sign) based on a particular α-shape chosen as a "good" linear approximation to the surface to be reconstructed. The computation of the signed distance is then reduced to locating the query point q in both the Delaunay triangulation, to decide its sign $s = \pm 1$, and in the Voronoi diagram, to find the closest point $p \in P$. The signed distance $s \cdot |pq|$ is then returned.

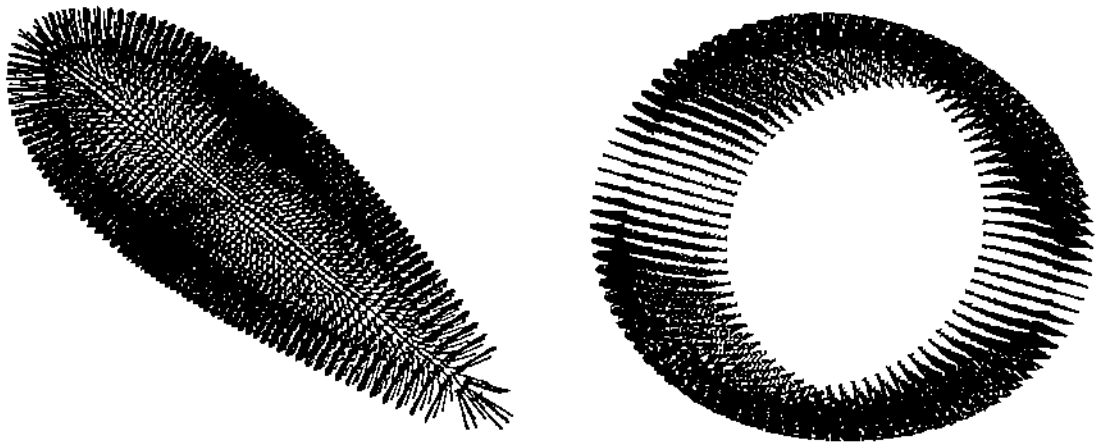
See Figures 4.1, 4.2, 4.3 and 4.4, showing the result of the α shape construction for various point sets and Figure 4.5 for the correct orientation in the sign distance computation.

4.3 Density of Points and Suitable α

A difficulty in the process outlined above is the choice of a suitable value for α. We assume that the input data is dense enough so that there exists an α so that the α-shape approximates the object with the same topology as the orginal unknown surface S. By this we mean that there exists a value for α such that the corresponding α-shape Σ_α contains only simplices that lie in the interior or on the boundary of the object, and that the outermost shell of the alpha shape (the subcomplex of Σ_α formed by vertices, edges and triangles lying on the boundary of the object) has the following properties:

(a) it does not contain any singular (i.e. isolated) vertex;

(b) there are no "missing" edges, i.e.for any two triangles $\tau_1, \tau_2 \in T$ that lie on the boundary of the object (notice that τ_1, τ_2 may or may not be in Σ_α) their common face (an edge) belongs



(a) (b)

Figure 4.5: Orientations showing the Correct Computation of Signed Distances for a Jet Engine Nose and Outer Cowl

to Σ_α.

In our current scheme a suitable α is selected interactively. When an α-shape with the above properties is determined, it is easy to distinguish between interior and exterior tetrahedra in the underlying triangulation T. One does a breadth first search on the dual graph of T, starting with a tetrahedron that is known to be external (e.g. one that has a vertex at infinity) and continuing with adjacent tetrahedra. These tetrahedra are marked as exterior (sign $-$) and put in a queue for further processing. When one hits a tetrahedron τ belonging to Σ_α, τ is marked as interior and not enqueued. The same happens when, visiting an adjacent tetrahedron τ of a negative tetrahedron σ one finds that the adjacency face belongs to Σ_α. This means that going from σ to τ one crosses the boundary, so τ is marked as interior and not enqueued.

4.4 Trivariate Interpolation Scheme

Steps 7 and 9 of the algorithm require a C^1 cubic local interpolation scheme using C^1 data at the vertices of the tetrahedral mesh. Both the domain surface piecewise cubic polynomial $F_T^{(1)}$ and the surface-on-surface piecewise cubic polynomial $F_T^{(2)}$ are computed using the same trivariate interpolation scheme.

We base our trivariate scheme on the n-dimensional Clough-Tocher scheme given by Worsey and Farin [34]. In this scheme, each tetrahedron is split into twelve subtetrahedra by choosing a point in the tetrahedron and a point on each face. A cubic trivariate interpolant is bulit on each subtetrahedron. This splitting step can also be found in [19] however insufficient details are provided in both [19, 34] for computations of the coefficients of the cubic interpolant. We now provide details of the steps for determining the coefficients of the split cubic interpolant over a macrotetrahedron.

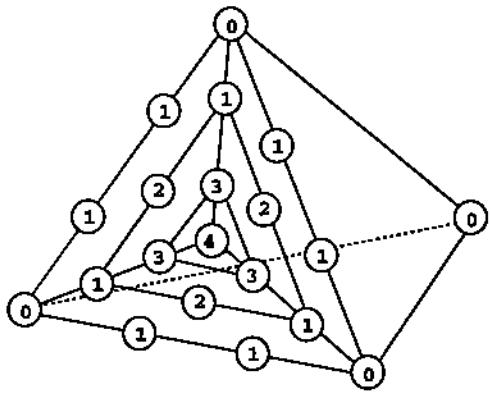


Figure 4.6: The Bezier ordinates on the cubic shell

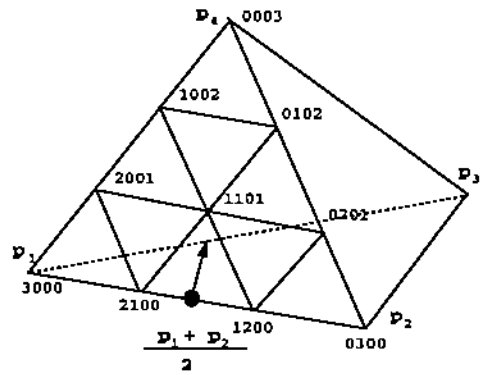


Figure 4.7: b_{1101} is determined from its known neighbors and directive derivative

Let us call the outer face of the tetrahedron a cubic shell. If we peel off the cubic shell, we are left with a quadratic shell. If we repeat this peeling what remains is a linear shell and the end a center point. See Figures 4.6, 4.7 and 4.8. The coefficients of the cubic trivariate interpolant are determined in the same manner, from the outer to the center.

On the cubic shell. On each face, the coefficients are computed as two dimensional Clough-Tocher scheme(see Figure 4.6). That is

1. The weights labeled 0 are obtained from the function values at the vertices.
2. The weights labeled 1 are computed from the gradient values at the vertices using Lemma 2.1.
3. The weights labeled 2 are computed from the gradient at the mid-edge points. Now we show the computation of b_{1101}(see Figure 4.7). Let

$$n_{124} = (p_4 - p_1)a_{124} + (p_4 - p_2)b_{124}$$

with

$$a_{124} = (p_1 - p_2)^T(p_2 - p_4), \quad b_{124} = (p_2 - p_1)^T(p_1 - p_4)$$

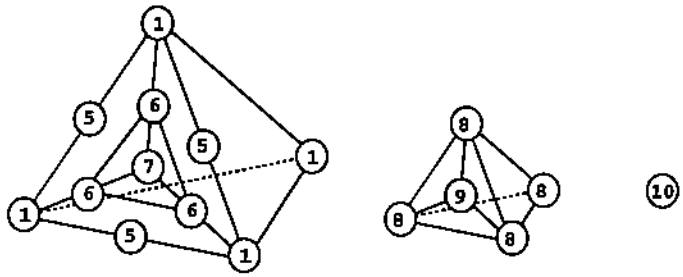


Figure 4.8: The Bezier ordinates on the quadratic shell, the linear shell and the center point

Then n_{124} is the normal of the line $\langle p_1p_2\rangle$ in the plane $\langle p_1p_2p_4\rangle$. Then by Lemma 2.3, the directional derivative in the non-normalized direction n_{124} of the cubic $f(p) = F(\alpha)$ at the point $p_{12} = \frac{p_1+p_2}{2}$ is

$$\begin{aligned}\nabla f(p_{12})^T n_{124} &= \nabla f(p_{12})^T(p_{12} + n_{124} - p_{12}) \\ &= \tfrac{3}{4}[(a_{124}+b_{124})b_{0201} - a_{124}b_{1200} - b_{124}b_{0300}] \\ &+ \tfrac{3}{2}[(a_{124}+b_{124})b_{1101} - a_{124}b_{2100} - b_{124}b_{1200}] \\ &+ \tfrac{3}{4}[(a_{124}+b_{124})b_{2001} - a_{124}b_{3000} - b_{124}b_{2001}]\end{aligned}$$

It follows that

$$\begin{aligned}b_{1101} &= \frac{\frac{4}{3}\nabla f(p_{12})^T n_{124} + a_{124}(b_{3000}+b_{2100}) + b_{124}(b_{0300}+b_{1200})}{2(a_{124}+b_{124})} \\ &+ \frac{b_{2100}+b_{1200}-b_{2001}-b_{0201}}{2}\end{aligned}$$

All the other number 2 weights are obtained by symmetry.

4. The weights labeled 3 are computed from their surrounding number 1 and number 2 weights by Lemma 2.2.

5. The weights labeled 4 are computed from their surrounding number 3 weights by Lemma 2.2.

On the quadratic shell. These coefficients are determined by the given data and C^1 condition(see Figure 4.8). That is

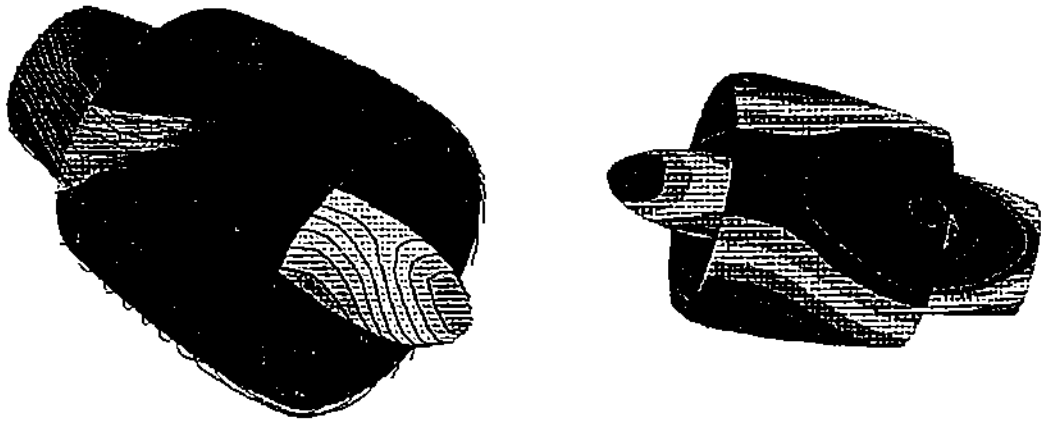
1. The weights labeled 1 are computed from the gradient at the vertices, using Lemma 2.1.

2. The weights labeled 5 are computed from their surrounding number 1 and number 2 weights on the cubic shell by Lemma 2.2.

3. The weights labeled 6 are computed from their surrounding number 1 and number 5 weights on the current shell by Lemma 2.2.

4. The weights labeled 7 are computed from their surrounding number 6 weights by Lemma 2.2.

On the linear shell.



(a) (b)

Figure 5.1: Iso-Pressure Contours of a Surface-on-Surface Pressure Function displayed on the Surface of the Jet Engine

1. The weights labeled 8 are computed from their surrounding number 1 and number 5 weights on the quadratic shell by Lemma 2.2.

2. The weights labeled 9 are computed from their surrounding number 8 weights by Lemma 2.2.

On the center point.

1. The weights labeled 10 are computed from their surrounding number 8 weights on the linear shell by Lemma 2.2.

Another trivariate Clough-Tocher scheme (see [1]) splits each tetrahedron into four subtetrahedra. However the interpolants in each subtetrahedra are now of quintic degree and furthermore require C^2 data at the vertices of the main tetrahedron. Since our data at the vertices of the tetrahedral mesh comes from the averaging of locally computed low degree interpolants, the higher order derivatives tend to be un-reliable in general. We therefore prefer to use the lower degree cubic scheme that uses only first order derivatives at the vertices.

5 Visualizing the Surface-on-Surface Function

We now present techniques for different visualizations of the domain surface $S_d : F^{(1)}(x, y, z) = 0$ and the surface-on-surface $S_f : F^{(2)}(x, y, z) = W$. Remember that S_f is a two dimensional surface in $\mathbb{R}^4$ since its domain is the two dimensional surface S_d.

5.1 Iso-Contours on the Surface

One approach is to visualize the curve iso-contours $W = constant$ of the surface S_f on the domain surface S_d by showing different colors in the region between two iso-contours. We achieve this

by first generating a triangle approximation of the domain surface S_d using adaptive contouring [29], and correspondingly generate triangles on $F^{(2)}$ in $\mathbb{R}^4$ (easy since this is a function). Next we intersect these triangles on $F^{(2)}$ with iso-values $W = constant$ to get a linear approximation of the curve iso-contours. For our implementation we do the following. Let w be a given iso-value, $[p_1p_2p_3]$ be a triangle on S_d. Without loss of generality, we may assume $F^{(2)}(p_1) \leq F^{(2)}(p_2) \leq F^{(2)}(p_3)$. Then if $w < F^{(2)}(p_1)$ or $w > F^{(2)}(p_3)$, the triangle does not intersect the iso-value. If $w \in [F^{(2)}(p_1), F^{(2)}(p_3)]$, say $w \in [F^{(2)}(p_1), F^{(2)}(p_2)]$, let

$$t_1 = \frac{w - F^{(2)}(p_1)}{F^{(2)}(p_2) - F^{(2)}(p_1)}, \quad t_2 = \frac{w - F^{(2)}(p_1)}{F^{(2)}(p_3) - F^{(2)}(p_1)}$$

$$q_1 = t_1p_1 + (1 - t_1)p_2, \quad q_2 = t_1p_1 + (1 - t_2)p_3$$

then $[q_1q_2]$ is one segment of the curve iso-contour $F^{(2)}(p) = w$. The collection of all of these line segments form a piecewise linear approximations to the iso-contours. By increasing the resolution of the triangulation of the domain surface S_d, we can obtain better linear approximations of the iso-contours for smooth looking displays. See Figure 5.1 where such an approach has been followed for the jet engine and pressure scalar function visualizations.

5.2 Surface-on-Surface in 3D

Since the curve iso-contours may not clearly indicate the geometric shape of the surface, one often likes to display the projection of the surface-on-surface from $\mathbb{R}^4$ to $\mathbb{R}^3$. One approach is to use a radial projection from the center of the domain surface S_d. However, if the domain surface S_d is not convex or has non-zero genus, this method has serious difficulties. Another more natural way is to use the normal projection, that is, project the point p on the domain surface S_d to a distance proportional to $F^{(2)}(p)$ in the normal direction of S_d at p:

$$G(p) = p + L\frac{\nabla F^{(1)}(p)(F^{(2)}(p) - {F^{(2)}}_{min})}{\|\nabla F^{(1)}(p)\|({F^{(2)}}_{max} - {F^{(2)}}_{min})}$$

where L is a positive scaler, ${F^{(2)}}_{min}$ and ${F^{(2)}}_{max}$ are minimum and maximum values of $F^{(2)}$ on S_d. However again, if L is not given properly, the surface G may self-intersect in case the domain surface S_d is not convex.

To avoid self-intersections we take $L < \gamma = \min_{p \in S_d}\{R_{min}(p)\}$, where $R_{min}(p)$ is the minimal principal curvature radius at p of surface S_d. Since S_d is a piecewise C^1 continuous surface and each piece is C^∞, hence $\gamma > 0$. The main task here is to compute γ. In general, it is not easy to compute the exact value of γ, however an approximate γ serves our purpose as well. When we produce the triangulation of each surface patch on S_d, we also compute $R_{min}(p)$ for vertices p, and then take $\gamma = \min_{p \in S_d}\{R_{min}(p) : p \text{ is a vertex}\}$. For an implicit surface $f(x_1, x_2, x_3) = 0$, let $f_i = \frac{\partial f}{\partial x_i}$, $f_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$. The principal curvature radius are the absolute values of the roots of the equation $ax^2 + bx + c = 0$, with $a = \sum_{i=1}^3 f_i^2(f_{i+1,i+1}f_{i+2,i+2} - f_{i+1,i+2}^2) + 2\sum_{i \neq j, k \neq i, k \neq j} f_if_j(f_{ik}f_{jk} - f_{ij}f_{kk})$, $b = \|\nabla f\|(\sum_{i=1}^3 f_i^2(f_{i+1,i+1} + f_{i+2,i+2}) - 2\sum_{i \neq j} f_if_jf_{ij})$, and $c = \|\nabla f\|^4$ where the indices are taken mod 3. Another approach to determine L is to take

$$L < \delta = \min_{p \in T_d}\left\{\min_q\left\{\frac{\|q - p\|^2\|\nabla F^{(1)}(p)\|}{2(q - p)^T\nabla F^{(1)}(p)} : [p, q] \in T_d \text{ and } (q - p)^T\nabla F^{(1)}(p) > 0\right\}\right\}$$

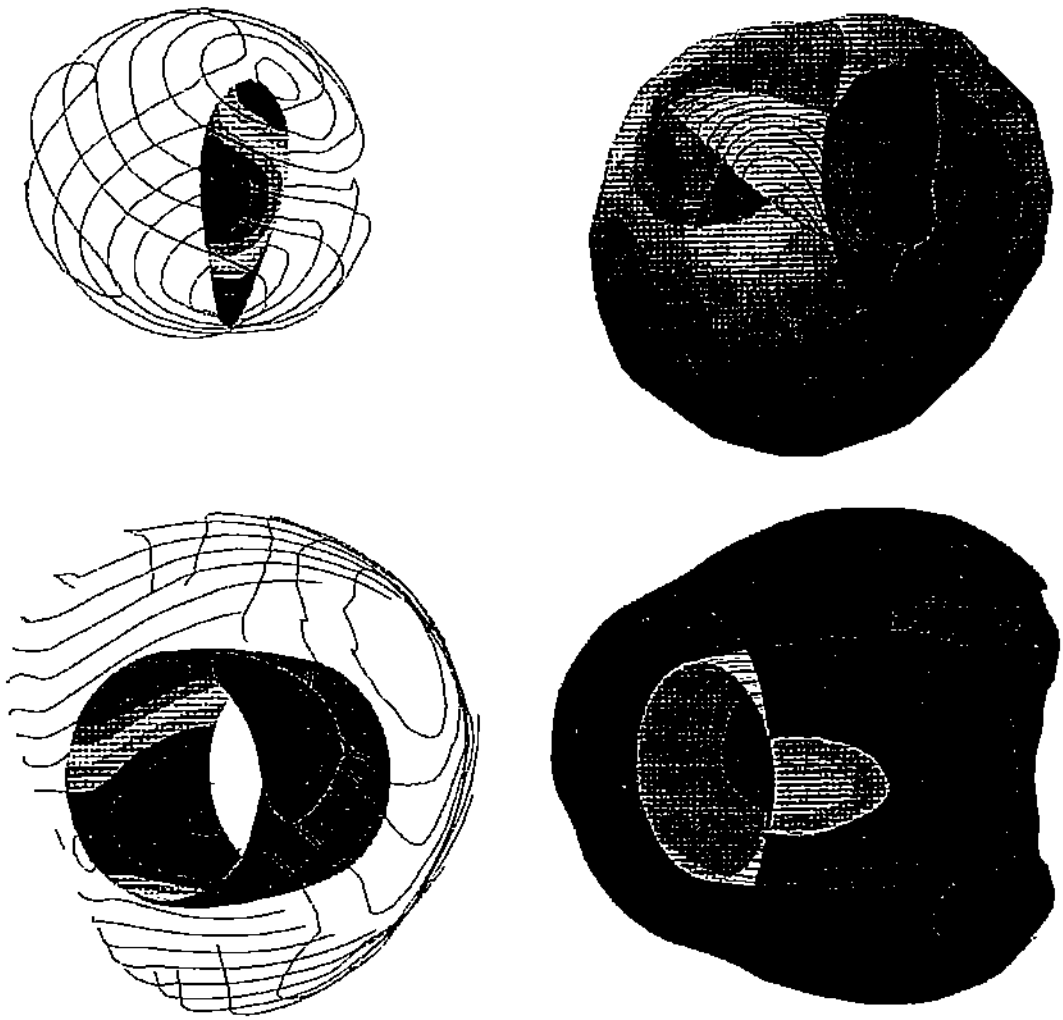


Figure 5.2: Iso-Pressure Contours of a Surface-on-Surface Pressure Function and Visualization of the Pressure Surface Function Surrounding the Nose and Outer Cowl of the Jet Engine using the Normal Projection method

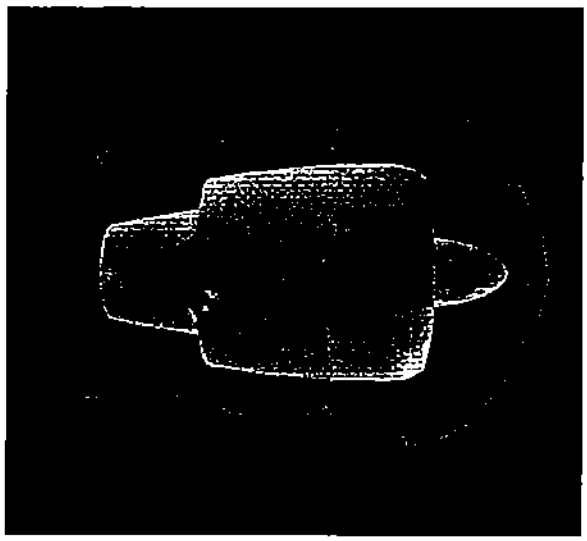


Figure 5.3: The Reconstructed Engine Surface and Visualization of the Pressure Surface Function Surrounding the Jet Engine using the Normal Projection method

where T_d is the triangulation of S_d used to display S_d;, $p \in T_d$ means p is a vertex and $[p,q] \in T_d$ means $[p,q]$ is an edge. It should be noted that if S_d is convex(i.e., $(q-p)^T \nabla F^{(1)}(p) \leq 0$) or planar(i.e., $(q-p)^T \nabla F^{(1)}(p) = 0$), the δ can be chosen arbitrarily. See Figures 5.2 and 5.3 where the visualizations have been constructed by the normal projection method detailed above.

Acknowledgements: We thank Herbert Edelsbrunner for several enlightening discussions on the topic of weighted alpha shapes.

References

[1] P. Alfeld. A trivariate clough-tocher scheme for tetrahedral data. *Computer Aided Geometric Design*, 1:169–181, 1984.

[2] P. Alfeld. Scattered Data Interpolation in Three or More Variables. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 1–34. Academic Press, 1989.

[3] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surveys*, 23:345–406, 1991.

[4] C. Bajaj, J. Chen, and G. Xu. *Interactive Modeling with A-Patches.* Computer Science Technical Report, CSD-TR-93-002, Purdue University, 1993.

[5] C. Bajaj and I. Ihm. C^1 Smoothing of Polyhedra with Implicit Algebraic Splines. *SIGGRAPH'92, Computer Graphics*, 26(2):79–88, 1992.

[6] R. Barnhill. Surfaces in computer aided geometric design : A survey with new results. *Computer Aided Geometric Design*, 2:1–17, 1985.

[7] R.E. Barnhill and T.A. Foley. Methods for constructing surfaces on surfaces. In G.Farin, editor, *Geometric Modeling: Methods and their Applications*, pages 1–15. Springer, Berlin, 1991.

[8] R.E. Barnhill, K. Opitz, and H. Pottmann. Fat surfaces: a trivariate approach to triangle-based interpolation on surfaces. *Computer Aided Geometric Design*, 9:365–378, 1992.

[9] R.E. Barnhill, B.R. Piper, and K.L. Rescorla. Interpolation to arbitrary data on a surface. In G.Farin, editor, *Geometric Modeling*, pages 281–289. SIAM, Philadelphia, 1987.

[10] J. Boissonnat. Representing 2D and 3D Shapes with Delaunay Triangulation. In *Proc. of the 7th Intl. Conference on Pattern Recognition*, pages 745–748, 1984.

[11] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comp.*, 17(41):830–847, August 1988.

[12] W. Dahmen. Smooth piecewise quadratic surfaces. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 181–193. Academic Press, Boston, 1989.

[13] W. Dahmen and T-M. Thamm-Schaar. Cubicoids: modeling and visualization. *Computer Aided Geometric Design*, 10:93–108, 1993.

[14] T. Dey, C. Bajaj, and K. Sugihara. On Good Triangulations in Three Dimensions. In Proc. of the ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, pages 131–141, Austin, Texas, 1989.

[15] H. Edelsbrunner. Algorithms in Combinatorial Geometry. Springer Verlag, 1987.

[16] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Comput. Sci. Dept., Univ. Illinois, Urbana, Ill., 1992.

[17] H. Edelsbrunner, D.G. Kirkpatrick, and H. Seidel. On the shape of a set of points in the plane. *IEEE Trans. on Information Theory*, 29:4:551–559, 1983.

[18] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. In *Proc. of 8th Ann. Sym. Comp. Geom.*, pages 43–52, 1992.

[19] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3:83–127, 1986.

[20] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1990.

[21] T.A. Foley, D.A. Lane, G.M. Nielson, R. Franke, and H. Hagen. Interpolation of scattered data on closed surfaces. *Computer Aided Geometric Design*, 7:303–312, 1990.

[22] R. Franke. Recent advances in the approximation of surfaces from scattered data. In C.K.Chui, L.L.Schumarker, and F.I.Utreras, editors, *Multivariate Approximation*, pages 275–335. Academic Press, New York, 1987.

[23] B. Guo. Surface generation using implicit cubics. In N.M. Patrikalakis, editor, *Scientific Visualizaton of Physical Phenomena*, pages 485–530. Springer-Verlag,Tokyo, 1991.

[24] B. Guo. Non-splitting Macro Patches for Implicit Cubic Spline Surfaces. *Computer Graphics Forum*, 12(3):434 – 445, 1993.

[25] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[26] D. Moore and J. Warren. Approximation of dense scattered data using algebraic surfaces. In *Proc. of the 24th Hawaii Intl. Conference on System Sciences*, pages 681–690, Kauai, Hawaii, 1991.

[27] G. Nielson, T. Foley, B. Hamann, and D. Lane. Visualizing and Modeling Scattered Multivariate Data. *IEEE Computer Graphics And Applications*, 11:47–55, 1991.

[28] G.M. Nielson. Scattered Data Modeling. *IEEE Computer Graphics And Applications*, 13:60–70, 1993.

[29] C. S. Peterson. Adaptive Contouring of Three Dimensional Surfaces. Computer Aided Geometric Design, 1(00):61–74, 1984.

[30] H. Pottmann. Interpolation on surfaces using minimum norm networks. *Computer Aided Geometric Design*, 9:51–67, 1992.

[31] Preparata, F., and Shamos, M. *Computational Geometry, An Introduction.* Springer Verlag, 1985.

[32] K.L. Rescorla. C^1 Trivariate Polynomial Interpolation. *Computer Aided Geometric Design*, 4:237–244, 1987.

[33] R. Veltkamp. 3D Computational Morphology. *Computer Graphics Forum*, 12(3):115 – 127, 1993.

[34] A. J. Worsey and G. Farin. An n-dimensional clough-tocher interpolant. *Constructive Approximation*, 3:99–110, 1987.