Optimal Linear Spline Approximation of Digitized Models

Bernd Hamann[1], Oliver Kreylos[1], Giuseppe Monno[2] and Antonio E. Uva[2]

[1]*Center for Image Processing and Integrated Computing (CIPIC)*
Department of Computer Science, University of California, Davis, USA
hamann@cs.ucdavis.edu, okreylos@gallagher.cipic.ucdavis.edu

[2]*Dipartimento di Progettazione e Produzione Industriale*
Politecnico di Bari, Bari, Italy
gmonno@poliba.it, uva@dppi.poliba.it

Abstract

In this paper we present a new technique for surface reconstruction of digitized models in three dimensions. Concerning this problem, we are given a data set in three-dimensional space, represented as a set of points without connectivity information, and the goal is to find, for a fixed number of vertices, a set of approximating triangles which minimize the error measured by the displacement from the given points.

Our method creates near-optimal linear spline approximations, using an iterative optimization scheme based on simulated annealing. The algorithm adapts the mesh to the data set and moves the triangles to enhance feature lines. At the end, we can use the approach to create a hierarchy of different resolutions for the model.

1. Introduction

Surface reconstruction is concerned with the extraction of shape information from point sets. Often, these point sets describe complex objects and are generated by scanning physical objects, by sampling other digital representations (e.g., contour functions), or by merging data from different sources. The result of this scanning process is usually a cloud of points at a very high resolution but without connectivity information. In order to utilize this data for actual modeling in a CAD system, it is important to reduce the amount of data significantly and determine a polygonal representation from the samples. Moreover, multiple approximation levels are often needed to allow rapid rendering and interactive exploration of massive data sets of this type. Surface reconstruction problems arise in a wide range of scientific and engineering applications, including reverse engineering, industrial design, geometric modeling, grid generation, and multiresolution rendering.

1.1. Related Work

Hoppe et al.[4] address the problem of reconstruction of surfaces using only the three-dimensional coordinates of the data points. Their method uses a "zero-set" approach to reconstruction, using the input points to create a signed distance function d, and then triangulating the isosurface d=0. They determine an approximate tangent plane at each sample point, using a least-squares approximation based on k neighbors. The isosurface is then generated using the marching cubes algorithm.

Amenta[14] directly uses a three-dimensional Voronoi diagram, and an associated Delaunay triangulation to generate certain "crust" triangles which are used in the final triangulation. The output of their algorithm is guaranteed to be topologically correct and convergent to the original surface as the sampling density increases.

Heckel et al.[2] introduce a surface reconstruction method that is based on cluster analysis. The reconstructed model is generated in two steps. First, an adaptive clustering method is applied to the data set, which yields a set of almost flat shapes, so-called tiles. Second, the gaps are eliminated between the tiles by using a constrained Delaunay triangulation, producing a valid geometrical and topological model. This method allows one to create a hierarchy of representations.

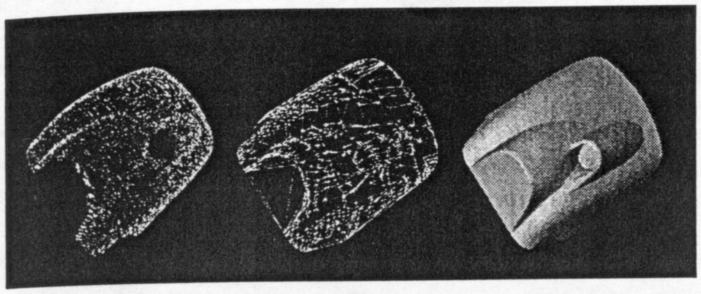
1.2. Our Approach

We present a new "optimal" (more precisely, near-optimal) method for the generation of surface triangulation. Our method exhibits the following characteristics:

- It requires only scattered points in the space, without connectivity information;

0-7695-0210-5/99 $10.00 © 1999 IEEE

- it needs a minimal user-interaction for a general topology, none for particular topologies;
- it generates an optimal approximation of the surface, with a fixed number of vertices; and
- it produces a multiresolution approximation of the data, where the user can specify the number of vertices in the reconstruction.



(a) The original point data set. (b) The final triangulation. (c) The shaded reconstructed model. The original data set consists of 37,594 sample points and the model has been reconstructed with 400 vertices.

Figure 1. Laser scan of a Ski-Doo hood.

The algorithm reconstructs a valid triangulated surface model in a three-step procedure:

- **Cutting** the data set in topological simple areas
- **Optimization** applied to all areas
- **Stitching** the shells together

2. Cutting Step

The core reconstruction algorithm treats the point cloud as a set of samples of a two-dimensional function f(x,y), where the samples are taken at random sites (x_i, y_i). If the original surface, or the surface we want to reconstruct, is not functional, the algorithm will deliver invalid results. This forces us to first find a mapping M from three-dimensional space (x,y,z) into two-dimensional parameter space (s, t) such that the mapping satisfies

$$(x, y, z) = M^{-1}(s, t, f(s, t))$$

for a function f. If we restrict ourselves to use orthogonal projections onto planes to define such a mapping, we have to find a plane the surface can be projected onto without self-overlap. If the model is too complex to find such a projection plane, we subdivide the model into smaller parts of simpler topology and provide different projection planes for the parts. To achieve this we first visualize the cloud of points. The user interacts with "cutting planes" subdividing the data set in sub-domains. Instead of cutting planes we should more precisely say "half-spaces" since we visualize the planes and the oriented normals. All the points included in those half-spaces are mapped (using orthogonal projection) onto the respective planes. Since the result of this subdivision is quite hard to visualize, the convex hull of the projections on the plane is shown to a user. This subdivision phase is typically not necessary if we are given a set of laser scanned images. This type of device usually captures points as distance from a sensor and then geometrically evaluates the xyz-coordinates. For all the points coming from a single pass scan, we are sure to find one single orthogonal projection plane.

3. Optimization Step

For each of the sub-domains (where all the points are "functional" in the way described above), we apply our iterative optimization algorithm based on the principle of simulated annealing, see [8][9][10]. The core of this algorithm is a function that changes the current triangulation randomly in every iteration step. After each step, a distance between the triangulation and the original data set is calculated, and the current step is accepted or rejected based on the change of distance during the step. The main difference between simulated annealing algorithms and classic optimization algorithms is that a simulated annealing algorithm not only accepts "good" steps, but also accepts some steps that increase the distance.

The strategy to accept steps is borrowed from thermodynamics, where Boltzmann's law states that a change ΔE in internal energy of a body occurs with the probability:

$$p = e^{-\frac{\Delta E}{kT}}$$

where k is the Boltzmann constant, and T is the absolute temperature of the body. We use the same function here, replacing ΔE with the change in distance during a step and kT with an arbitrary value we nonetheless call "temperature." If ΔE is negative, meaning the step was a good step, we always accept it; in the other case, we accept it with the probability given by Boltzmann's law.

We then lower the value kT during the course of iteration to decrease the probability of accepting "bad" steps. In the end, when the temperature is almost zero, the algorithm proceeds like a classic optimization. The function decreasing the temperature over time is called "annealing schedule," and [1] presents a heuristic to create it. The benefit of allowing "bad" steps is, that such algorithms do not as easily get stuck at local minima as classic algorithms do. This is an important property for us, since we are dealing with problems typically having local minima in abundance, see Figure 2.

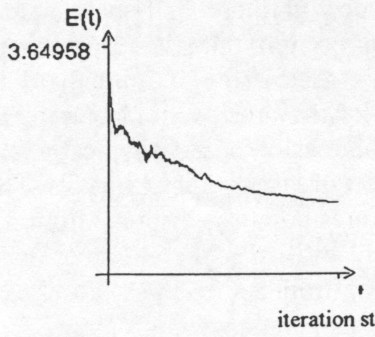


Figure 2. Typical error graph: general error behavior as a function of iteration steps.

The previous formula shows the effect of the temperature on the probability of accepting a bad step.

The user can define the number of vertices to be used in the triangular approximation of the sub-domains. The following pseudo-code summarizes the optimization algorithm, which is described in more detail in the following sections.

Algorithm 1: Optimal linear spline approximation.

```
Create initial configuration (vertex placement and
  connectivity);
Determine initial temperature and create annealing
  schedule;
While iteration is not finished
  {
  Change current configuration;
  Calculate change in error measure;
  Undo iteration if rejected by simulated annealing;
  }
Return current configuration;
```

3.1. Creating an Initial Configuration

To evaluate a reasonable initial configuration we start to determine the data set's convex hull by selecting all non-interior vertices; then we choose the rest of the vertices (according to the user-specified number) randomly from the original data set. A Delaunay triangulation of the initial vertices' sites defines the initial connectivity.

To define the annealing schedule, we first estimate the mean change in distance during the first iteration steps and set the initial temperature such, that an "expected bad" step is initially accepted with a probability of one half. Next, we lower the temperature in steps, leaving it constant for a fixed number of iterations and scaling it by a fixed factor afterwards.

3.2. Changing the Configuration

The simulated annealing algorithm's core is its iteration step. In principle, one can use any method to change the current configuration, but we have found out that the "split" approach, presented in Algorithm 2, works very well.

Algorithm 2: Changing the configuration.

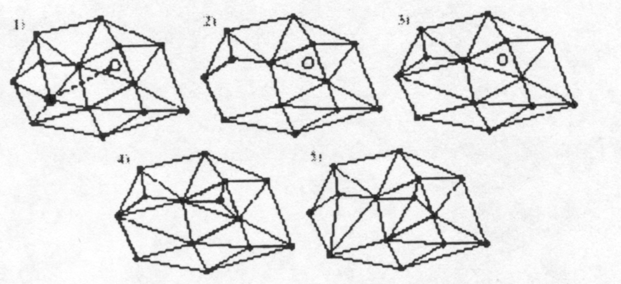
```
if(acceptWithProbability(moveVertex)) /*move a vertex*/
  {
  Choose an interior vertex v;
  Estimate v's contribution vE to the error measure;
  if(vE < localMovementFactor × E)
      Move v globally;
  else
      Move v locally;
  if(moveVertex == 1) /* Vertex movements only?*/
      Restore Delaunay property;
  }
else /* swap an edge */
  {
  Choose a swappable edge e;
  Swap edge e;
  }
```

The constant ***moveVertex*** gives the probability of moving a vertex during an iteration step. If it is zero, the algorithm never moves vertices, but becomes a data-dependent triangulation algorithm as presented in [8]. If ***moveVertex*** is one, we only move vertices, and we decided to uphold the connectivity's Delaunay property throughout the iteration in this case. In all other cases the algorithm can either move a vertex or swap an edge, thereby optimizing both vertex placement and triangulation simultaneously. When moving a vertex, we use two different strategies:

- If the chosen vertex is located in a planar region of the surface, we move it **globally** to a random new position inside the point set's convex hull, see Figure 3.
- If the chosen Vertex is located in a high-curvature region of the surface, we move it **locally** to a random new position inside its platelet, see Figure 4.

We decide which strategy to use by estimating how much the chosen vertex contributes to the current distance. If this contribution is larger than a constant factor ***localMovementFactor*** times the distance, we move the vertex globally, otherwise, we move it locally.

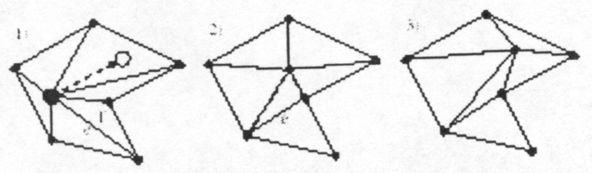
By using global movements we ensure that vertices get driven away from nearly planar regions of a function during early stages of the iteration.



1) Initial state; 2) removing vertex; 3) filling hole; 4) inserting new vertex; 5) restoring Delaunay property.

Figure 3. Moving a vertex globally.

If the vertex is currently located in an important high-curvature region of the surface, we keep this vertex "in loco" and we try to move it to a better site inside its platelet. To move a vertex **locally**, we "slide" the vertex on the line from its old to its new site, dragging the edges connecting it to all surrounding vertices along. Whenever a surrounding simplex becomes degenerate during the vertex' motion, we swap one edge of the affected simplex before moving the vertex any further, see Figure 4.



1) Initial state; 2) swapping edge to prevent triangle T from becoming degenerate; 3) resulting state.

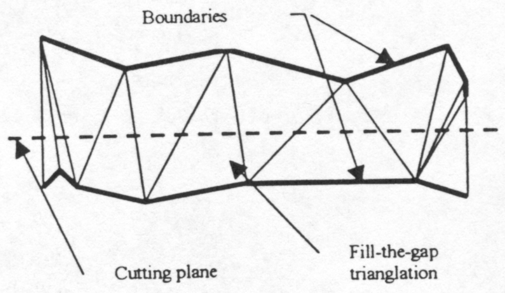
Figure 4. Moving a vertex locally.

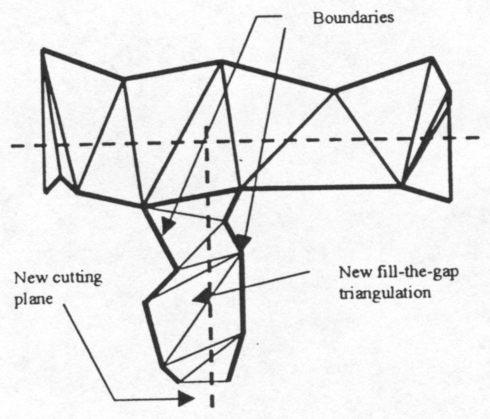
In our case, the quality of a configuration depends on both vertex placement and connectivity. The output of this second step is a set of two-manifold shells, which correspond to the number of topological sub-domains the whole data set was subdivided into. Defining an increasing number of vertices, we can create a hierarchy of linear spline approximations, each one being a superset of all lower-resolution ones (Figure 6.)

4. Stitching Step

In this step we stitch together the boundaries of the shells to define a consistent model. We do not move the vertices of the shell boundaries to match, but we add a new set of fill-the-gap triangles. The input of the stitching algorithm consists of two boundaries, each of them described by a sequence of vertices and edges. To define the boundaries we first project each point into the selected half-space onto the cutting plane. Then we calculate the convex hull boundary on the plane and afterwards map the segments back into three-dimensional space. The second boundary is evaluated with the same procedure with a few modifications for the unselected half-space. In this case not all the points participate in the definition of the convex hull.

The stitching algorithm is applied every time a cut is performed. In this way we already have two sets of points (and edges) to define a strip to be triangulated. We start from a random point and we find the closest point on the other boundary. Those two points define a new edge. For each vertex we maintain a flag indicating whether this vertex was already matched or not. We repeat this edge-creation step until all the vertices are matched. Then we scan the edge list to eliminate duplicates and we extract the fill-the-gap triangles (Figure 5.a.) When a new cut is performed over a previous subset, an edge of its boundary (defined by the previous cut) is hit by the cutting plane. This edge is preserved in the subsequent phase to ensure that this new fill-the-gap set will match exactly on the boundary (Figure 5.b.)





(a) First stitch. (b) subsequent stitch preserving the first triangulation.

Figure 5. Stitching two boundaries.

5. Results

This method has revealed to be very powerful with regard to error reduction. Concerning the Ski-Doo test case, after a few thousand iteration steps (requiring just a few seconds on an SGI Octane), we reduced the error to 40% of the initial configuration error (Delaunay triangulation). Figure 6 shows how the vertices move away from flat areas (large triangles) to converge towards the features of the model (high curvature, small triangles). The last example shows the reconstruction of a model with 2,000 vertices starting from a 1,2000 points data set (Figure 7).

6. Acknowledgements

This work was supported by various grants and contracts awarded to the University of California, Davis, and to the Politecnico di Bari, Italy, including the National Science Foundation under contract ACI 9624034 (CAREER Award), the Office of Naval Research under contract N00014-97-1-0222, the Army Research Office under contract ARO 36598-MA-RIP, the NASA Ames Research Center under contract NAG20-1216, the Lawrence Livermore National Laboratory under contract W-7405-ENG-48 (B-335358), the Department of Energy as part of the Accelerated Strategic Computing Initiative (ASCI) under contract W-7405-ENG-48, the North Atlantic Treaty Organization (NATO) under contract CRG. 971628, and the National Research Council (CNR), Italy. We also acknowledge the support of Silicon Graphics, Inc. We thank all members of the Visualization Thrust at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and of the Dipartimento di Progettazione e Produzione Industriale, Politecnico di Bari, Italy, for their help.

7. References

[1] Kreylos, O., Hamann, B., "On simulated annealing and the construction of linear spline approximations for scattered data", *EUROGRAPHICS-IEEE TVCG Symposium on Visualization*, Vienna, Austria, May 1999 (to appear).

[2] Heckel, B., Uva, A. E., and Hamann, H., "Clustering-based generation of hierarchical surface models, " in: Wittenbrink, C. M. and Varshney, A., eds., Late Breaking Hot Topics Proceedings, *Visualization '98* (Research Triangle Park, North Carolina, October 1998), IEEE Computer Society Press, Los Alamitos, California, pp. 41-44.

[3] Bonneau, G. P., Hahmann, S. and Nielson, G. M., "BLaC-wavelets: A multiresolution analysis with non-nested spaces", in Yagel, R. and Nielson, G. M., eds., *Visualization '96* (1996), IEEE Computer Society Press, Los Alamitos, CA, pp. 43-48.

[4] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., "Surface reconstruction from unorganized points," in Computer Graphics (SIGGRAPH '92 Proceedings), pp. 71-78, 1992.

[5] Gieng, T. S., Hamann, B., Joy, K. I., Schussman, G. L. and Trotts, I. J., "Constructing hierarchies for triangle meshes", *IEEE Transactions on Visualization and Computer Graphics* 4(2) (1998), pp. 145-161.

[6] Hamann, B., "A data reduction scheme for triangulated surfaces", *Computer Aided Geometric Design* 11(2) (1994), pp. 197-214.

[7] Hamann, B., Jordan, B. J. and Wiley, D. A., "On a construction of a hierarchy of best linear spline approximations using repeated bisection", *IEEE Transactions on Visualization and Computer Graphics* 5(1) (1999).

[8] Schumaker, L. L. "Computing Optimal Triangulations Using Simulated Annealing", *Computer Aided Geometric Design* 10 (1993), pp. 329-345.

[9] Nielson, G. M., "Scattered data modeling", *IEEE Computer Graphics and Applications* 13(1) (1993), pp. 60-70.

[10] Press, W. H., Teukolsky, S. A., Vetterling, W.T., and Flannery, B. P., *Numerical Recipes in C*, 2nd ed. (1992), Cambridge University Press, Cambridge, MA.

[11] Barequet, G., Duncan, C.A., and Kumar, S., "RSVP: A geometric toolkit for controlled repair of solid models", *IEEE Trans. on Visualization and Computer Graphics* (TVCG), vol. 4 (2), pp. 162-177, April-June 1998.

[12] Barequet, G. and Sharir, M., "Filling gaps in the boundary of a polyhedron", *Computer Aided Geometric Design* 12(2), pp. 207-229, 1995.

[13] Gueziec, A., Taubin, G., Lazarus, F., Horn, W., "Converting Sets of Polygons to Manifold by Cutting and Stitching", in Yagel, R. and Nielson, G. M., eds., *Visualization '98* (1998), IEEE Computer Society Press, Los Alamitos, CA, pp. 43-48.

[14] Amenta, N., Bern, M., Kamvysselis, M., "A new Voronoi-based surface reconstruction algorithm", *Siggraph '98*, (1998), pp 415-421.

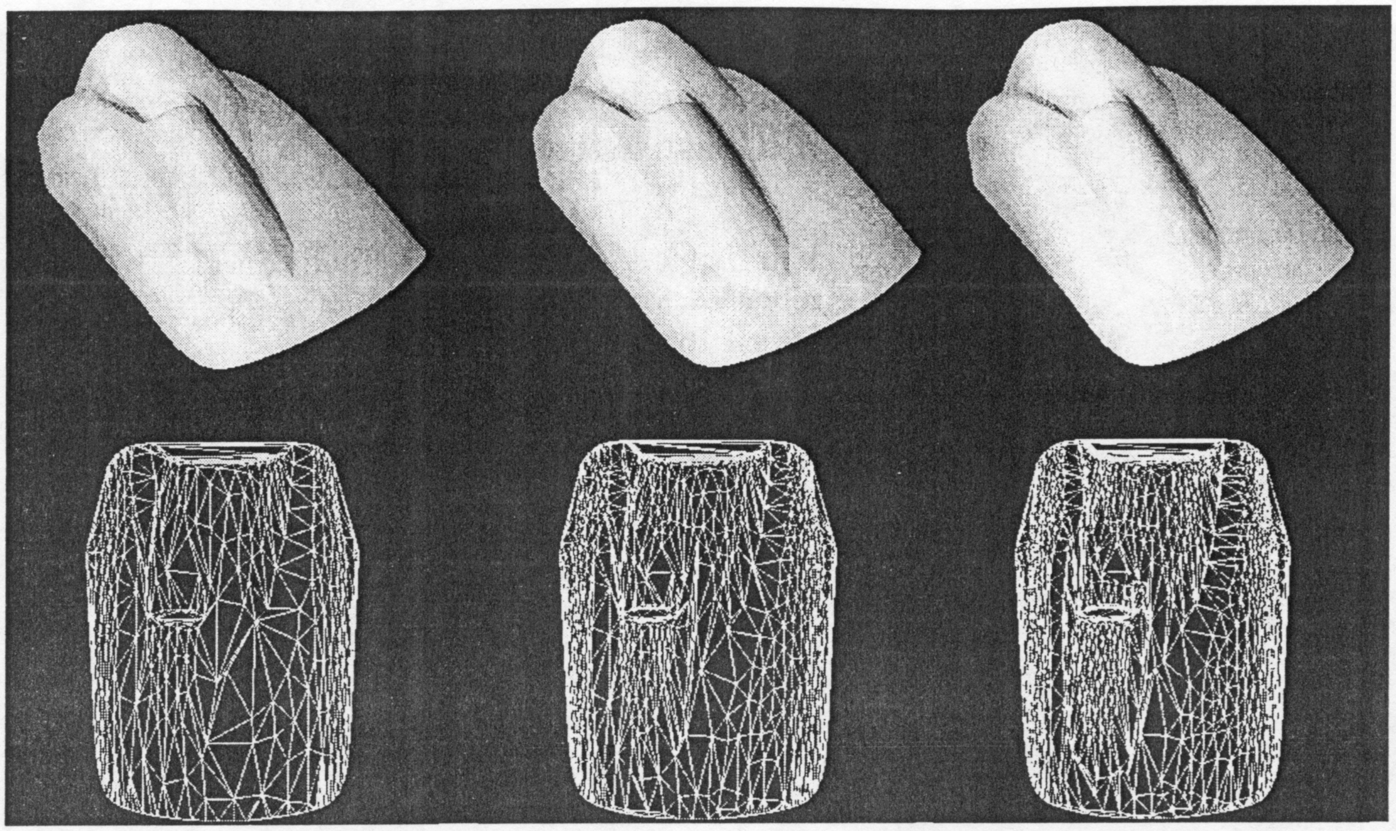
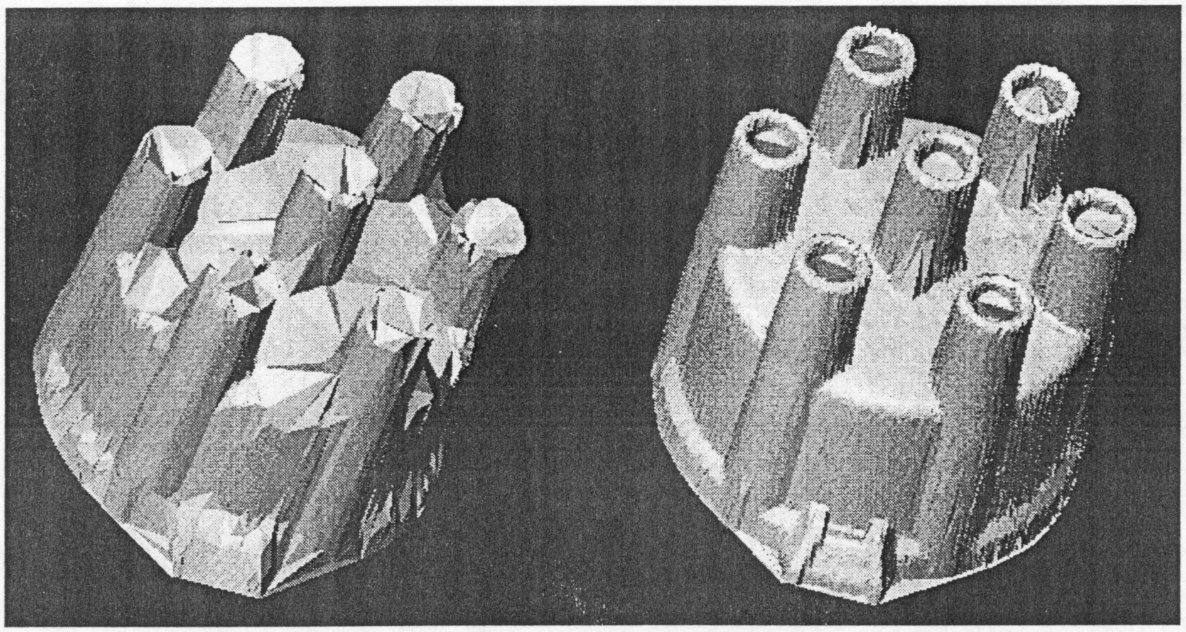


Figure 6. Multiresolution reconstructions using 400, 700, and 1000 vertices for the Ski-Doo data set.



a) Randomly chosen points b) After optimization step

Figure 7. Reconstruction of a mechanical part using 2,000 vertices for a 12,000 points data set.