Reconstruction of Surfaces from Scan Paths

Claus-Peter Alberts

Department of Computer Science VII,
University of Dortmund
D-44227 Dortmund, Germany
alberts@ls7.cs.uni-dortmund.de

Abstract. Given a finite set of points sampled from a surface, the task of surface reconstruction is to find a triangular mesh connecting the sample points and approximating the surface. The solution presented in this paper takes into consideration additional information often available in sample data sets in form of scan paths. This additional information allows to cope with edges or ridges more reliably than algorithms developed for unstructured point clouds.

1 Introduction

Let us given a set of scan paths covering a surface. The task is to create a triangular mesh which includes the points of the scan paths and approximates the surface.

Figure 1 shows a cloud of points sampled from a surface in which scan paths can be recognized. A scan path is a sequence of neighboring points which are for instance obtained by moving the scanning sensor along a smooth curve over the surface and sampling its location at a certain rate. We assume that the sampling points are linearly arranged in the input data set according to their occurrence on the scan paths. However, we do not assume to have the information about locations at which the scan process is interrupted in order to move the sensor to a new location where scanning continues with a new scan path. Data sets of this type are typically delivered by tactile scanners like the Cyclon [13].

Figure 2 shows a part of a triangular mesh reconstructed from the point set of Fig. 1. Most of the vertices of the mesh are points of the input sampling set, but some additional points, often called Steiner points in literature, may occur, too. In a general setting, the construction of a triangular mesh representing the surface topologically and geometrically correctly is a non-trivial task. A considerable number of approaches have been suggested, many of which are compiled in the survey by Mencl and Müller [10]. We assume that the surface can be described by a function over a plane, in the following chosen as the x-y-plane. Under this condition a triangulation can be obtained by triangulating the flat set of points obtained by orthogonally projecting the sampling points onto this plane. However, although the problem of finding the correct topology is solved by this assumption, it turns out that such triangulations are not always geometrically

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 295–304, 2002.

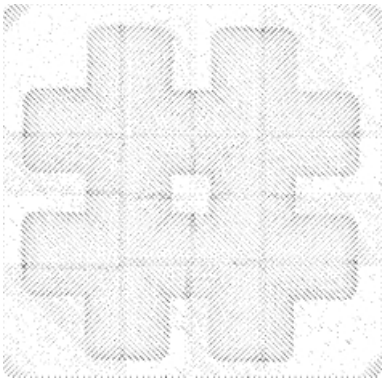

Fig. 1. Sample points of an object.

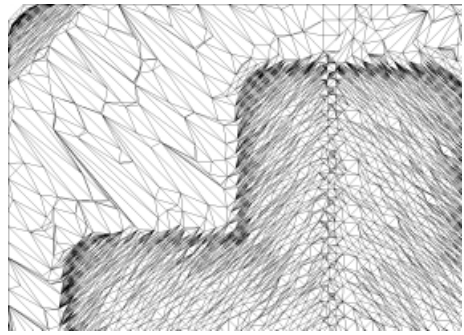

Fig. 2. Part of a triangulation.

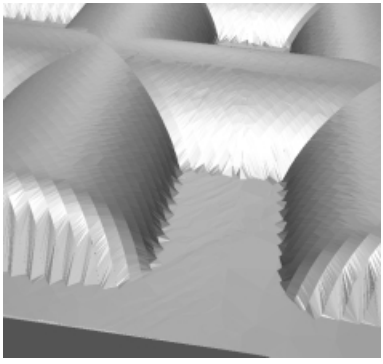

Fig. 3. The Delaunay triangulation.

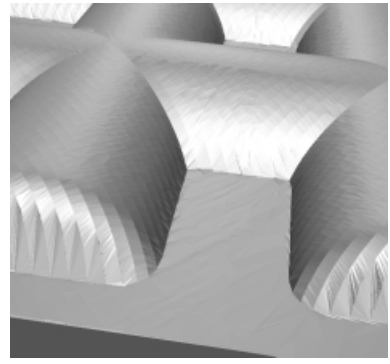

Fig. 4. The path adaptive triangulation considering feature trees.

satisfactory if the height information (z-coordinate) is not taken into consideration. Figures 3 and 4 illustrate the difference between a non-favorable and a favorable triangulation. Several data-dependent triangulation schemes have been proposed in the past which take into account spatial information [3, 4] (Fig. 5 and 6). A major criterion is curvature. In particular, a surface might have sharp edges or ridges which need special treatment like demonstrated for a different kind of sampling by [9]. In this case a typical approach is to decompose the surface into segments which are free of those artifacts, find a smooth triangulation of each of those segments, and compose them into a complete solution [7].

The solution to the surface reconstruction problem presented in this paper takes into consideration the additional information present in the sampling data set in form of scan paths. This additional information allows to cope with edges or ridges more reliably than algorithms developed for unstructured point clouds. For instance, consecutive points of a scan path should usually be connected by an edge because it can be expected that an edge of this type is close to the

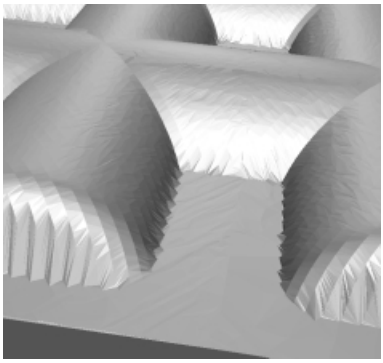

Fig. 5. The ABN triangulation.

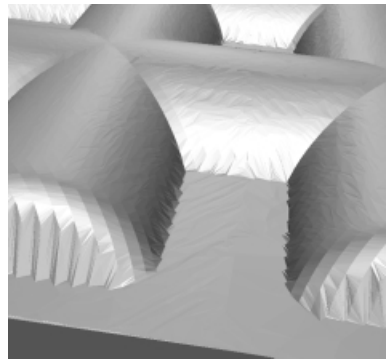

Fig. 6. The TAC triangulation.

surface. Furthermore, the relative dense points on a scan path are favorable to find points of high curvature which are characteristic for edges and ridges. Those points can be connected by edges to a polygonal path which represents a ridge, and thus should also be part of the triangulation.

The rest of this paper is organized along the steps of our approach. The first step, scan path extraction from the data set, is outlined in Sect. 2. The second step is filtering in order to reduce noise typical for scanned data (Sect. 3). The third step is to detect shape features along a scan path (Sect.4). In the fourth step an initial triangulation is determined which takes the scan paths into consideration (Sect. 5). In the fifth step, feature paths are calculated from the feature points of step 4 and are worked into the triangulation (Sect. 6).

2 Path Extraction

First we have to discover the start and end points of the scan paths. We use a heuristics that assumes some kind of fuzzy collinearity of the scan paths. This is a reasonable assumption for two reasons. Firstly, scan paths are often indeed linear in their projection on the x-y-plane. Secondly, for samplings of high density, even on a curved path two consecutive line segments are almost collinear.

Two consecutive points p_i and p_{i+1} of the input sequence are considered to belong to the same path if either

(1) the projected distance of p_i and p_{i+1} in the x-y-plane is sufficiently small, or
(2) the distance is large but the line segment $\overline{p_i p_{i+1}}$ is a fuzzy collinear continuation of $\overline{p_{i-1} p_i}$.

The angle between the two consecutive line segments under which they are still accepted as "collinear" depends on the length of $\overline{p_i p_{i+1}}$. It is specified by a function like that one of Fig. 7 which assigns the largest acceptable angle to an edge length.

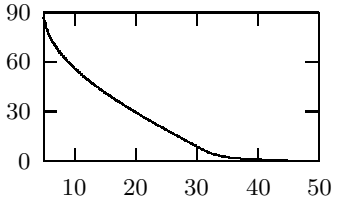

Fig. 7. Angles (vertical axis) accepted by the path extraction depending on the distance of two consecutive points (horizontal axis).

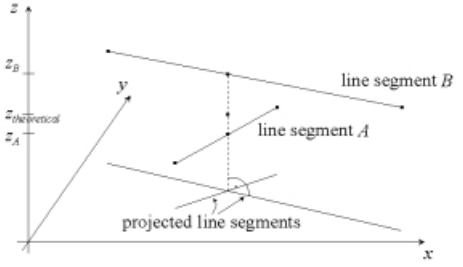

Fig. 8. Two crossing line segments.

The set of edges of the selected scan paths are checked for intersection in the projection on the x-y-plane (Fig. 8). For every pair of intersecting edges the z-coordinates z_A and z_B of the corresponding points of the intersection point on both edges are calculated. They are compared with a z-value $z_{theoretical}$ which is calculated as an average with weights reversely proportional to the edge lengths. If the difference exceeds a threshold the correlated edge is removed, that is a scan path is split into two paths. The background of this heuristics is that in this case longer edges cannot be a good approximation of the surface. They are removed preferably because it is more likely that short edges belong to a scan path.

3 Filtering

Sampled data usually are noisy because of mechanical or electrical properties of the sampling device. Typically the noise is high-frequent. High-frequent noise can be removed by low pass filters. The danger, however, is that sharp bends are smoothed out, too. We try to diminish this effect by using two different filters, one for smoothing the points according to their x- and y-coordinates, and one for smoothing the z-coordinate.

For filtering the x-y-coordinates, we assume the scan paths to be approximately straight-line except at sharp bends, that is at points where a path changes its direction significantly.

In order to maintain the sharp bends we use an adaptive Laplacian filter:

$$filter_{al}(p_i) = \mu_i \cdot \frac{1}{2}(p_{i-1} + p_{i+1}) + (1 - \mu_i) \cdot p_i.$$

An example of a non-adaptive Laplacian filter which sets μ_i, $0 \leq \mu_i \leq 1$, to constant value can be found in [14]. "Adaptive" means that the original point p_i is considered in a way that the larger a bend at p_i is the less it will be moved. μ_i depends on the angle the line segments incident to p_i include, on the property

whether there is a sharp bend or not (within the x-y-plane) at p_i, and on the distance between the examined points [1].

Although we limit the possible movement of a point p_i, there is still the possibility that p_i is moved too far. In order to avoid this we introduce a second filter, the adaptive limited gravitation filter:

$$filter_{alg}(p_i) = \vartheta_i \cdot \frac{1}{2}(p_{i+1} + p_i + \nu_i(p_{i-1} - p_{i+1})) + (1 - \vartheta_i) \cdot p_i.$$

$\vartheta_i, 0 \leq \vartheta_i \leq 1$, models "adaptivity" and is calculated similar to μ_i. $\nu_i, 0 \leq \nu_i \leq 1$, is a measure for the attraction of the points p_{i-1} and p_{i+1} to p_i. It is reverse proportional to the length of the incident line segments to p_i. A constant value would smooth out flanks, especially if the filter is applied iteratively. The attraction avoids spreading out an area with many points which indicate significant changes.

The complete xy-filter is the average of these two filters.

The filter for the z-values should handle two situations. On the one hand it should smooth noisy parts and on the other hand it should keep sharp bends and slopes. For the first purpose a low pass filter seems to be appropriate while for the second purpose the median filter seems to be suitable [5]. So we use an adaptive combination of them:

$$filter_z(z_i) = \zeta_i filter_{lowpass}(z_i) + (1 - \zeta_i) filter_{median}(z_i),$$

with ζ_i, $0 \leq \zeta_i \leq 1$, measuring the difference between $filter_{lowpass}(z_i)$ and $filter_{median}(z_i)$.

4 Shape Features

The shape features of the scan paths we are interested in are sharp bends. In order to find sharp bends it is sufficient to investigate the z-profile and ignore the behavior of a path with respect to the x- and y-coordinates. We call the points representing sharp bends within a path "(sharp) feature points".

We use three methods to detect sharp bends properly which result in different valuations of every point p_i, the "angle valuation", the "compensation valuation", and the "curvature valuation". These valuations are aggregated and then representative points are selected. The three methods have different weaknesses and strengths. The combination just described yields a better overall result.

The first way of detecting a sharp bend at a point p_i is to use the angle α (Fig. 9) between the two line segments incident to p_i. A real number $0 \leq r_{angle} \leq 1$ is calculated which is based on α and indicates the possibility of a sharp bend at p_i. Because of improper scanning it is possible that the actual sharp bend lies between p_i and p_{i+1}. Therefore a second number $0 \leq r_{2^{nd}angle} \leq 1$ is calculated based on the angle γ (Fig. 9). Then $r_{2^{nd}angle}$ is shared between p_i and p_{i+1}. The "angle valuation" of p_i is chosen as the maximum of r_{angle} and $r_{2^{nd}angle}$.

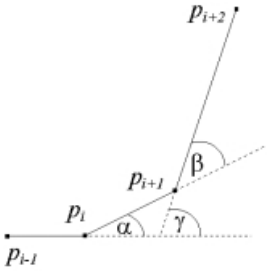

Fig. 9. Angles for determination of sharp bends within scan paths.

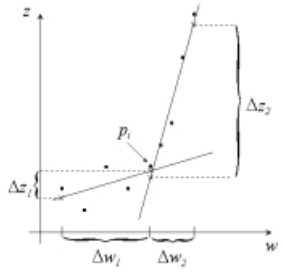

Fig. 10. Example of lines fitted into the profile of a path.

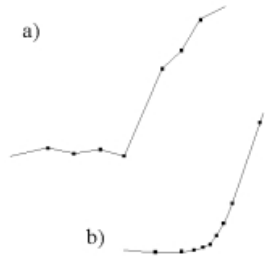

Fig. 11. Two different types of sharp bends.

In order to level out slight unevenness we calculate a further valuation of p_i. Therefore we take two regression lines through a number of points, one line compensating for unevenness before and one line compensating for it after p_i (Fig. 10). Then the "compensation valuation" is chosen as a function that maps the angle between these lines onto the real interval $[0, 1]$.

The last method detecting a sharp bend is to take the curvature of a path profile into account. We just calculate the curvature on the lines fitted into the profile of a path (see compensation valuation) because the curvature on the points without compensation is too sensitive to slight unevenness. The curvature k of a point p_i is measured by

$$k = \frac{\frac{\Delta z_1}{\Delta w_1} - \frac{\Delta z_2}{\Delta w_2}}{\sqrt{(\Delta w_1 + \Delta w_2)^2 + (\Delta z_1 + \Delta z_2)^2}}$$

with $w_i = \sum_{k=r_j+1}^{i} \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}$ if p_{r_j} is the start point of the containing path and z_i is the z-coordinate of p_i in the profile of this path. For the definition of the variables see Fig. 10. The definition of k is a modified version of that in [8] which favors special kinds of profiles unsuitably [1]. Then the "curvature valuation" is chosen as a function which maps k onto $[0, 1]$.

In order to calculate a single value for each point p_i the three valuations must be combined. We do this in fuzzy logic manner [6] by evaluating the phrase "a sharp bend is at $p_i \iff$ the angle valuation *or* the compensation valuation *or* the curvature valuation gives a hint". In order to get a value of the left side of the arrow we have to aggregate the values on the right side. For our purposes we have chosen the "algebraic or" $(vel_a(r, s) = r + s - r \cdot s)$.

In this manner sharp bend a) in Fig. 11 will be detected properly, but sharp bend b) will get just a small valuation. The reason is that many points get a medium valuation instead of one point getting a big one. Because of our observation, we give every point the chance to increase its valuation taking the points in its neighborhood into account.

This procedure results in several points with more or less high valuations. Because a sharp bend can be represented by only one point, we have to select a representative. This is done by choosing the point with the highest valuation in a selection interval. Not only the valuation of a point but also its z-coordinate is considered by the selection since the sharp bend b) in Fig. 11 leads to a feature point with a relative high z-coordinate otherwise. If there is more than one point with maximal valuation and there is sufficient space between the left- and rightmost point both are selected as representatives.

5 Triangulation

Because we assume that scan paths describe the surface of a scanned object properly we are interested in triangular meshes which comprehend the scan paths. We call a triangulation with this property "path-adaptive triangulation". Using our special configuration of functional surfaces we basically can construct an approximating triangular surface mesh by triangulating the projected sampling points in the x-y-plane and then connecting two original points if they are adjacent in the triangulation in the x-y-plane. However, path-adaptation makes the procedure somewhat more complicated.

The first step of construction of a path-adaptive triangulation is to split path edges intersecting in their projection on the x-y-plane, by insertion of additional points. The reason is that intersecting edges are forbidden in a triangulation. The additional points are the intersection points of the projected edges. The pairs of intersecting edges are efficiently found by first testing the line segments of every path for intersection. This test is implemented by processing the projected edges in lexicographically sorted order of their vertices. The edges of different paths are tested by processing the axes-parallel bounding boxes of the projected paths in lexicographical order of their vertices with the goal of finding the pairs of intersecting bounding boxes. For the pairs of intersecting boxes the pairs of intersecting edges are determined by processing the edges in lexicographical order. In this manner practical computation times are achieved for data sets with hundred thousands of points. Furthermore, the procedure is easier to implement than well-known worst-case efficient algorithms of computational geometry [11].

The second step is to calculate a first triangulation of the projected sampling points and the projected additional points of the first step. We use the classical Delaunay triangulation for this purpose [11]. We calculate the Delaunay triangulation by sorting the points lexicographically according to their x- and y-coordinates, combining the first three points into a triangle, and then successively connecting the other points to the convex hull of the already inserted points, in sorted order. If necessary, edges are flipped until they satisfy the empty circle condition [2]. Edge flipping means to replace an edge e with the diagonal different from e in the quadrilateral defined by the two triangles incident to e (if it exists), if it is inside the quadrilateral.

The third step is to convert the Delaunay triangulation into a path-adaptive triangulation by edge flipping. Let the edges of the triangulation be colored black

and those on scan paths not being part of the triangulation be colored red. The goal of edge flipping is to make the red edges to black edges, too. For this purpose we process the still existing red edges iteratively according to algorithm 1.

Algorithm 1. *Let $\mathfrak{L}$ be the set of edges crossing the red edge $\{p_s, p_e\}$. Furthermore, let $\mathfrak{L}$ be sorted according to the distance of the intersection points of its edges and $\{p_s, p_e\}$ to p_s.*
while $\mathfrak{L}$ *is not empty* **do**
if *there is a "flipable edge" in* $\mathfrak{L}$ **then** *flip it and remove it from* $\mathfrak{L}$ **else**
1. *Search* $\mathfrak{L}$ *in increasing order for a "forceable edge"* e_l.
2. *Flip* e_l *and successively all edges after* e_l *until a "flipable edge" or an "unforceable edge" is reached.*
3. *If an "unforceable edge" is reached then flip back the edges in reverse order except* e_l. *Flip successively all edges before* e_l *until a "flipable edge" or an "unforceable edge" is reached. If an "unforceable edge" is reached again then flip all edges back to their initial position (including* e_l*), look for a "forceable edge" after* e_l *and continue with step 2.*
4. *Flip the "flipable edge" and remove it from* $\mathfrak{L}$.

fi
od

Algorithm 1 uses three types of edges which are defined as follows: *flipable edges* do not intersect the red edge after flipping; *forceable edges* cross the red edge after flipping as well as before; *unforceable edges* have incident triangles forming a concave quadrilateral. After flipping unforceable edges would lie outside the quadrilateral, and therefore, they must not be flipped. This distinction is necessary because there are cases in which no flipable edges exist [1]. The existence of a forceable edge (if there is no flipable edge in $\mathfrak{L}$) and the correctness of algorithm 1, that means that the red edge is colored black in the end, have been proved in [1].

The average time of algorithm 1 seems to be quite difficult to estimate. Due to algorithm 1 every edge can be flipped at most three times within the steps 1 to 4. So the worst case is limited by $O(n^2)$ where n is the initial number of black edges in $\mathfrak{L}$.

6 Sharp Feature Trees

The next step is to construct curve-like sharp ridges of the scanned surface. For this purpose we use trees which connect feature points determined in previous steps of the algorithm. We call those trees "(sharp) feature trees". Up to now we do not consider closed paths because we have not yet developed criterions for closing a path to a loop.

A feature tree is obtained by building a minimum spanning tree (MST) in the triangulation, starting with an arbitrary feature point and using an algorithm similar to the MST-algorithm of Prim [12]. The algorithm of Prim takes one vertex as an MST and adds successively the point with the smallest distance

to all points of the MST to the present tree. Since feature points need not be adjacent in the triangulation (there might be gaps) and we calculate MSTs only on feature points we had to modify the algorithm of Prim using intermediate points. We also stop adding points to the MST if the distances of all vertices not yet in the tree to the vertices of the MST are too large. The details are given by algorithm 2.

Algorithm 2. *Let Q be a priority queue containing vertices and let Q be sorted according to the distance between the vertices and the corresponding feature point from which they have been found (possibly using intermediate points, see below). Furthermore let p_s be the initial feature point.*
Initialize Q with all points adjacent to p_s and close enough to p_s.
while *Q is not empty* **do**
  remove the first entry p_r of Q;
  if *p_r is not marked* **then**
    mark p_r
    if *p_r is a feature point* **then**
      add $\overline{p_r p_u}$ to the feature tree where p_u is the feature point from which p_r has been found (possibly using intermediate points);
    fi
    add all points to Q which are adjacent to p_r, not marked, and close enough to p_r or p_u, respectively;
  fi
od

If feature points not belonging to an MST are left, a further MST is established with one of those points as start point.

Finally the path-adaptive triangulation is adapted so that the feature trees become part of the triangulation. This is achieved by applying the edge flipping algorithm of Sect. 5.

7 Empirical Evaluation and Conclusions

The presented algorithm is a new approach to detect and consider sharp features for triangular meshes. Neither the Delaunay triangulation (Fig. 3) nor the ABN or TAC triangulation (Fig. 5 and 6) model sharp features as adequate as the presented triangulation (Fig. 4). This is not surprising since between 33.5 and 86.7% of the edges of the investigated feature trees were not part of the Delaunay triangulation.

Although the worst case of the run time of algorithm 1 might be quadratic, often linear run time was observed. Furthermore, in the special case we yield at it is not clear if an example can be constructed leading to the upper bound.

The computation time was up to 18 seconds for a data set with 33K points (Fig. 1) and up to 5 minutes for a data set with 220K points. We used a Pentium III with 500MHz and needed 64MB and 250MB RAM, respectively. Our algorithm has been implemented in JAVA2. Most of the computation time was

spent on calculating the Delaunay triangulation. The Delaunay, the ABN, and the TAC triangulation (Fig. 3, 5 and 6) were computed by another program which has been implemented in C++ in 3.5, 8.3, and 54 seconds, respectively.

One reason for smoothing paths is to get a better starting point for bend detection. However, it has a smoothing effect on the overall surface, too. Surface-oriented smoothing techniques might improve the result if necessary.

The algorithm has several control parameters. Experience shows that there are reasonable default values. Sometimes an improvement can be achieved by a data-adaptive choice. Currently the values are chosen interactively based on a visualization of the mesh. Automatic approaches to parameter optimization can be an interesting topic of future research.

More detailed discussions can be found in [1].

Acknowledgements.
I like to thank Prof. Heinrich Müller for scientific discussions.

References

1. Alberts, Cl.-P.: Triangulierung von Punktmengen aus Abtastdaten unter Berücksichtigung von Formmerkmalen. Masters Thesis, Department of Computer Science VII, University of Dortmund, Germany, 2001.
2. Bern, M.: Triangulations. In Goodman, J.E., O'Rourke, J. (editors): Handbook of Discrete and Computational Geometry. CRC Press, New York, 1997, 413–428.
3. Van Damme, R., Alboul, L.: Tight Triangulations. In Dæhlen, M., Lyche, T., Schumaker, L.L. (editors): Mathematical Methods for Curves and Surfaces. Vanderbilt University Press, Oxford, 1995, 185–192.
4. Dyn, N., Levin, D., Rippa, S.: Data Dependent Triangulations for Piecewise Linear Interpolation. IMA Journal of Numerical Analysis **10** (1990) 137–154.
5. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Addison-Wesley, 1992.
6. Gottwald, S.: Fuzzy Sets and Fuzzy Logic. Foundations of Application – from a Mathematical Point of View. Artificial Intelligence Series, Vieweg, 1993.
7. Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W.: Piecewise Smooth Surface Reconstruction. Proceedings of SIGGRAPH 1994, ACM Press, New York, 1994, 295–302.
8. Hoschek, J., Lasser, D.: Fundamentals of Computer Aided Geometric Design. A.K. Peters, Wellesley, MA, 1993.
9. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.-P.: Feature Sensitive Surface Extraction from Volume Data. Proceedings of SIGGRAPH 2001, ACM Press, New York, 2001, 57–66.
10. Mencl, R., Müller, H.: Interpolation and Approximation of Surfaces from Three-Dimensional Scattered Data Points. Proceedings of the Scientific Visualization Conference, Dagstuhl'97, IEEE Computer Society Press, 2000.
11. Preparata, F.P., Shamos, M.I.: Computational Geometry. An Introduction. Springer, New York, 1985.
12. Prim, R.C.: Shortest connection networks and some generalizations. Bell System Tech. Journal **36** (1957) 1389–1401.
13. http://www.renishaw.com/client/ProdCat/prodcat_levelthree.asp?ID=1229
14. Taubin, G.: A signal processing approach to fair surface design. Computer Graphics, Proceedings of SIGGRAPH'95, ACM Press, New York, 1995, 351–358.