Feature-line Extraction from Point Clouds*

Engelbert Westkämper; Ralph Peter Knorpp; Norbert Schuhmann

Abstract:
Feature lines are a substantial means for segmenting measured point clouds, especially in cases where a surface model needs to be generated from the data. In this article, methods will be described for the automated extraction of feature lines from measured or scanned point clouds. These methods can be applied to points from any sensors as they do not require sorted or edited data sets. In this paper, the mathematical basis for the extraction of feature lines, experimental results and the limitations of the methods will all be presented.

Keywords: *Computer Aided Engineering, Reverse Engineering, Feature Lines.*

Introduction

To date, a 3D CAD model is usually derived from an existing physical part firstly by scanning the physical part with a 3D scanner, e.g. laser triangulation scanner or coordinate measuring machine (CMM) and then measuring and analyzing the dense discrete point data in order to obtain the 3D CAD model. In the case of laser scanners with varying densities, for example, the measured data is typically in the form of large point clouds made up of millions of points. One class of reverse engineering methods is based on the segmentation of point clouds into polygonal regions to fit an individual surface to each region (s. [10]). To avoid undesired smooth surfaces in regions with sharp edges, the boundaries of each polygonal region should be formed by the sharp edges of the scanned part (together with additional boundaries). As this segmentation must be carried out manually in current CAD systems, this paper deals mainly with methods for extracting feature lines (sharp edges or edges with small radii) from scanned point clouds automatically. In order to be irrespective of special 3D scanners, no point cloud order information will be taken into consideration – the point cloud will be regarded as totally unsorted scattered data. Data structures will be presented capable of handling even huge point clouds efficiently.

The algorithms for automated feature-line extraction attempt to find locations where surface normal vectors suddenly change direction. This seems to be a contradiction as normal vectors cannot be determined until the surface has been computed, but normal vectors are required so that the boundaries of the surface may be calculated first. However, algorithms for the robust approximation of normal vectors have been developed and will be described in this paper. As the change in direction of normal vectors represents the curvature of the part surface, the feature line extraction will therefore be based on these point curvature values.

Data Structures for Operations on Dense Point Clouds

Scanning with modern tactile or optical devices results in point clouds containing as many as several millions of points. Algorithms depending quadratically on the number of points may no longer be used here due to the extended computing time. This chapter will present special data structures with acceptable runtimes which have been developed for handling operations on huge and dense point clouds. In the next chapter, algorithms for computing point curvature values will be specified. Point curvature values represent the curvature values of the part surface at each of the scanned points. Only the points immediately adjacent to each point are regarded, as only points in a relatively small neighborhood should influence these values. In order to determine all points located within a search radius R around a given point p_i of the cloud, the distances d_j of all other points p_j in the cloud around p_i are computed and checked to see whether $d_j < R$. If there are N points in the cloud, the computation of $N \cdot (N-1)$ distances is required, and this would result in the already-mentioned runtime problems for large point clouds.

To avoid this drawback, a special data structure is needed to compute only a few distances for each point. Several models have been tested and a special version of a 3D hashing table was used, usually known as a "3D grid method". A general description of the grid method can be found in [5, 7, 9].

The main concept of the grid method is to take only points within nearby grid cells into consideration. The grid cells are adjacent cubes with a constant size g and with axes parallel to the x-, y- and z-axes. The compound of all cubes enfolds the bounding box $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ of the point cloud. The last row in the x-, y-, and z-directions normally degenerates into a cuboid, as the quotient of the length of the bounding box $x_{max} - x_{min}$ (the other sides, respectively) and g is generally not an integer. The grid cells are numbered consecutively from x_{min} to x_{max} (the other sides, respectively) and an integer triple $\{ ?, \mu, ? \}$ is used to access the ?-th, μ-th and ?-th cell in the x-, y- and z-directions, which forms the cube in formula (1).

$$\begin{aligned}\{l,m,k\} \rightarrow & [x_{\min} + lg, x_{\min} + (l+1)g] \\ & \times [y_{\min} + mg, y_{\min} + (m+1)g] \\ & \times [z_{\min} + kg, z_{\min} + (k+1)g]\end{aligned} \quad (1)$$

The triple $\{ ?, \mu, ? \}$ represents the key for accessing the cells in the 3D hashing table. In order to calculate the key for each point p_i with the coordinates (x_i, y_i, z_i) of the cloud, formula (2) – (4) is used:

$$l := floor((x_i - x_{\min})/(x_{\max} - x_{\min})) \quad (2)$$

$$m := floor((y_i - y_{\min})/(y_{\max} - y_{\min})) \quad (3)$$

$$k := floor((z_i - z_{\max})/(z_{\max} - z_{\min})) \quad (4)$$

Here, the function $y=floor(x)$ returns the integer value y representing the largest integer, which is less than or equal to x. Applying these simple formulas (2) – (4) has the advantage that the key in the hashing table may be determined for each point by carrying out just a few arithmetic operations. In particular, the checking of every cell in the 3D grid, which would once again lead to major runtime problems can thus be avoided. Once the key for each point of the cloud has been determined, the neighbors of

* This work was supported by the European Community in the project "Intelligent Manufacturing Systems – Rapid Product Development", Brite-EuRam project BE-97-5501.

each point p_i can be established simply by searching within the cubes with a maximum distance away from p_i less than or equal to the search radius R. The size of the grid length g has a crucial influence on runtimes: If g is too small, too many cubes exist which contain no points and these are analyzed to no purpose. If g is too long, there are only a few cubes and these contain too many points. If no grid is used (which is the same as a grid with only one cube), runtime then veers towards the unacceptable, i.e. quadratic runtime. The diagram in fig. 1 shows the scaled runtime for various data sets containing approx. 100,000 points versus with the grid factor $gf := g / R$.

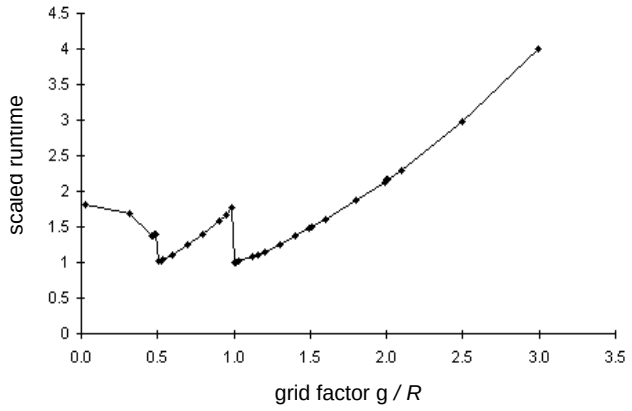

Fig. 1: Runtime vs. grid factor

Minimal runtime can be found for $gf \approx 0.5$ and $gf \approx 1.01$. Because the runtime for these grid factors does not differ much, the grid factor $gf \approx 1.01$ was chosen for determining the neighbors, as regarding the implementation fewer grid elements are needed for a higher grid length, thus leading to shorter access times.

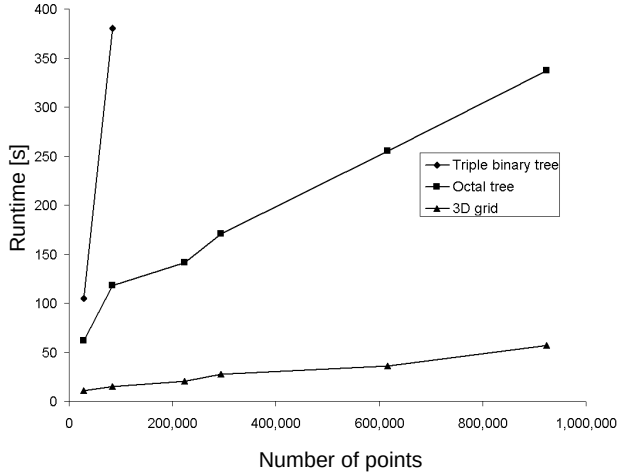

Fig. 2: Runtimes for the 3D grid, octal tree and triple binary tree methods

Two alternative data structures were tested and compared with the 3D grid method (s. fig. 2): the triple binary tree, consisting of three binary trees, one for each coordinate, and the octal tree. One point (x_i, y_i, z_i) of the cloud will be in each node of the octal tree, and the eight different sub-trees in every node represent the eight cases of the formula (5)-(7). The binary tree and the octal tree are not explained in detail here, as they are standard methods which can be found, for example, in [5, 7, 9].

$$1.\ x < x_i,\ y < y_i,\ z < z_i \tag{5}$$

$$2.\ x < x_i,\ y < y_i,\ z \geq z_i \tag{6}$$

$$\vdots$$

$$8.\ x \geq x_i,\ y \geq y_i,\ z \geq z_i \tag{7}$$

A runtime comparison of the three methods mentioned is given below which shows the runtime advantages using the 3D grid methods. The runtime of the methods mentioned also depends on the special implementation and the computer hardware used. An Intel 450MHz Pentium II single processor machine was used here with 128 MByte RAM.

Computation of Point Curvature Values

Once the neighbors of each point p_i of the point cloud within the search radius R have been determined, best fit planes E_i in each point depending only on the neighbors of p_i can be computed. With Taylor's theorem, the best fit plane E_i can be interpreted as being a local approximation of the surface near the point p_i. The normal vector of the plane E_i can then be interpreted as an approximation of the normal surface vector of the measured part. Fitting planes to point data is a well-known problem in CAD and is normally solved using least-squares methods. These methods are used here but are not explained in any detail (s. [2]).

Once the normal vectors n_j for the relevant points within the search radius R have been computed, point curvature values will be determined by applying formula (8) which gives an approximation of the average curvature of the surface of the part (s. [3]). To apply formula (8), the orientation of the normal vectors of the neighbored points must be homogenous. This must be ascertained first, because the calculation of point normal vectors with the best fit planes E_i does not automatically result in a homogenous orientation of the normal vectors (s. [4]).

$$c_i := \frac{1}{N_{nb}(p_i)} \sum_{j=1}^{N_{nb}} \left| \frac{n_i - n_j}{p_i - p_j} \right| \tag{8}$$

The homogenous orientation of two neighbored points p_i and p_j, may easily be checked if the distance between p_i and p_j is sufficiently small and the normal directions do not therefore vary significantly.

$$n_i \cdot n_j \approx +1 \Rightarrow n_i \text{ and } n_j \text{ are homogenously oriented} \tag{9}$$

$$n_i \cdot n_i \approx -1 \Rightarrow n_i \text{ and } n_i \text{ are not homogenously oriented}, \tag{10}$$

Here, $p_i \cdot p_j$ denotes the Euclidean scalar product of p_i and p_j. If the normal vectors are not homogenously oriented, n_j is multiplied with -1 to change its orientation.

Tracking Edges Based on Point Curvature Values

Using the point curvature values, a pre-segmentation of the points may be performed by applying a simple threshold operation. When implementing the algorithms, the point curvature threshold value is an interactive changeable value. In order to find an acceptable threshold value, all points with point curvature values higher than the threshold value are marked. Edge-tracking is carried out only on these pre-segmented points. Initially, the pre-segmented points are sorted by decreasing point curvature values. Starting with the point with the highest curvature value, the edges are then tracked. The main objective in tracking edges is to search

for points with maximum point curvature values. Once some edge points have been tracked, the neighbors of the last tracked edge point within a search radius R_t are determined using the 3D grid method explained above. Within the neighbor points, the point with the maximum curvature is searched for and stored as the next edge point. This procedure is repeated until no further edge point within the pre-segmented points can be found. (s. [1, 6, 8]).

If only these criteria are applied, the tracking of edges does not lead to the desired results, due to the following problems:

- The tracked edges will not cover the whole area of the points located with point curvature values above the threshold value. Due to the fact that results are sorted according to curvature values, above iteration would only track the first point with a maximum curvature value, and this would remain the point with the maximum curvature value for all further repeated steps. This must be avoided by excluding points which have already been tracked.

- If there is an area with points with relatively high curvature values, all these points would be tracked (sorted by decreasing curvature values), before the edge "leaves" this area and points outside would be tracked. Iteration would not lead to the desired result. In such cases, only a few representative points within this area should be tracked.

To avoid the problems mentioned above, forbidden areas need to be defined. Points within the forbidden areas are then excluded from the following iteration steps. Each time a new point is added to the edge list, the forbidden area is updated and enlarged. The forbidden area includes all points within a tube consisting of cylinders and spheres (s. fig. 3).

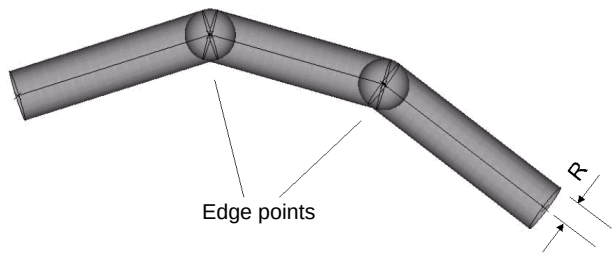

Fig. 3: Forbidden area made up of cylinders and spheres

For each edge line already detected, one cylinder is computed with this edge line as the centerline of the cylinder. For each edge point except the first and the last edge point, one sphere is computed with the edge point as the center of the sphere. The radius of the cylinders and the spheres is set to the search radius R_t.

Examples and Limitations of the Method

By applying the methods described here, it is clear that the quality of the tracked edges is dependent upon the quality of the point clouds measured. To measure noise dependency in a point cloud, a virtual point cloud consisting of 500 x 21 points is computed, superimposed with (computed) statistical noise. Here, normally-distributed noise value (Gaussian distribution) in x-, y- and z-directions is computed. The points are located on a regular grid on two intersecting planes (500 x 11 points each) with a defined angle of intersection superimposed with the computed errors. The deviation (Euclidean distance) between the computed feature-line and the original intersection line $dist_{fl}$ and also the standard deviation (RMS error) of the computed noise s are then divided by the average minimal distance $dist_{pt}$ of the points to obtain the scale invariant values for relative noise and relative deviation.

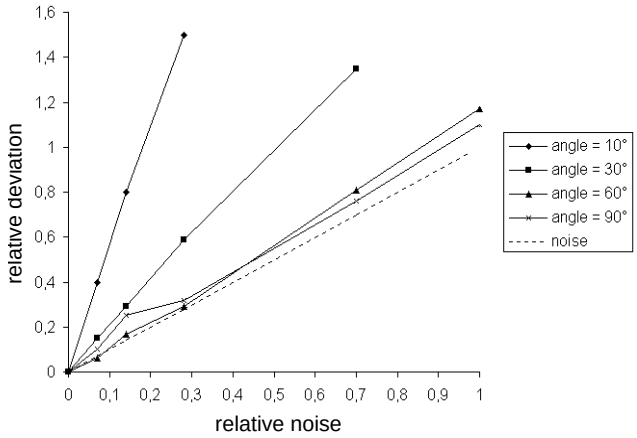

Fig. 4: Relative deviation of feature lines vs. relative noise (Gaussian distribution of errors)

Fig. 4 shows the dependency between the relative deviation of the computed feature line and the original intersection line vs. the relative noise for the different angles of intersection 10°, 30°, 60° and 90°. The diagram shows that the relative deviation decreases as the angles increase. As far as the intersection angles of 30°, 60° and 90° are concerned, the relative deviation is smaller than the relative noise, a fact which demonstrates the stability of these methods for noisy point clouds with sharp edges.

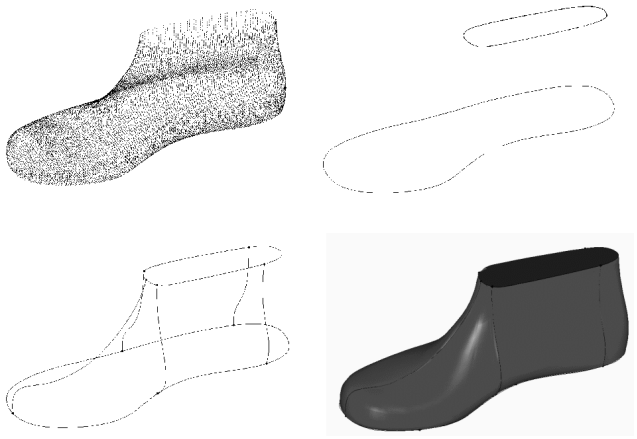

Fig. 5: Reverse engineering of a shoe: point cloud, extracted feature lines, completed boundaries, surface model

Fig. 5 shows four steps of the reverse engineering process for the test part, in this case a shoe: Scanned point cloud (top left), extracted feature lines (top right), feature lines completed with interactive constructed boundaries (bottom left) and surface model (bottom right). The new method has been tested extensively by industrial users. It showed that the manual extraction of feature lines without applying the new method is mostly a very time-consuming task. On the other hand, automated feature-line extraction usually takes just a few seconds. The accuracy of the feature lines extracted automatically also increases substantially in

comparison with manually extracted feature lines, as the quality of manually extracted feature lines is generally rather poor.

Further tests with scanned data showed that the deviation of the computed edges increases if the point cloud distribution is less uniform. With unevenly distributed clouds, the determination of neighbors for each point within a constant search radius results in point sets, with some of them possibly being too small for small search radii and some too big for big search radii. As a result, the computation of fit planes may either fail or be less accurate. This could occur, for example, in the case of triangulated point clouds which can be exported directly from some scanners. Before being exported, the triangulated point clouds are usually optimized by reducing triangles in flat areas of the scanned part in order to decrease the number of triangles and the resulting file size, which will give rise to unevenly distributed point clouds and the problems mentioned above.

Another possible problem is the existence of smooth edges, one example being the case where two intersecting planes have been trimmed and connected to a fillet with a constant or linearly-increasing radius. If the maximum radius of the fillet is relatively small, the described method results in edges “somewhere” in the middle of the fillet which may be considered as an acceptable result. However, in the case of fillets with larger radii where the algorithms listed here for the extraction of feature lines may possibly fail, an automatic detection of the trimming curves (beginning or end of the fillet, respectively) would be of more interest for reverse engineering applications. These fillet end lines are also called feature lines. The automated feature line extraction for non-uniform point distributions and the automatic detection of feature lines for fillets are currently being developed.

Conclusion

Automatic feature-line detection is an important tool for reverse engineering applications. Using the algorithms described, the location of sharp edges in 3D scanner data can be determined automatically. Data structures have been introduced which are capable of handling even huge, dense point clouds. The limitations of the methods are apparent in cases with non-uniform point distributions and parts with fewer sharp edges, but these problems are currently being worked on and will soon be ready for presentation.

References

1 *Bhandarkar, S. M.; Siebert, A.:* Integrating edge and surface information for range image segmentation. Pattern Recognition 25 (1992) 9, pp. 947-961

2 *Geise, G.; Schippke, S.*: Ausgleichsgerade, -kreis, -ebene, -kugel im Raum. Mathematische Nachrichten 62 (1974) pp. 65-75

3 *Hoffmann, R;, Jain, A.K.:* Segmentation and classification of range images. IEEE Transactions on Pattern Analysis and Machine Intelligence 9 (1987) No. 5, pp. 608-620

4 *Hoppe, H.:* Surface Reconstruction from Unorganized Points. PhD Thesis Univ. Washington, 1994

5 *Knorpp, R.*: Formleitlinien fuer die Flaechenrueckfuehrung – Extraktion von Kanten und Radiusauslauflinien aus unstrukturierten 3D-Messpunktmengen. Springer, Stuttgart 1998

6 *Lange, M.*: Segmentierung von Konturen auf der Basis von Kruemmungsberechnungen. 13. DAGM-Symposium Mustererkennung 1991. Radig, B. (Ed.).Springer, Berlin: 1991

7 Nievergelt, J.; Hinrichs, K.: Programmierung und Datenstrukturen. Springer, Berlin1986

8 *Roth, G.; Levine, D.*: Extracting geometric primitives. Computer Vision, Graphics Image Processing: Image Understanding 58 (1993), pp. 1-22

9 *Sedgewick, R.*: Algorithms. Addison-Wesley, New York 1991

10 *Trucco, E.; Fisher, R. B.:* Experiments in Curvature-Based Segmentation of Range Data. IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (1995) 2, pp. 177-182

11 *Varadi, T.; Martin R.; Cox J.*: Reverse engineering of geometric models – an introduction. In: Computer-Aided Design 29 (1997) 4, pp. 255-268

12 *Wang. W.; Iyengar, S. S.*: Efficient Data Structures for Model-Based 3-D Object Recognition and Localization from Range Images. IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (1992) 10, pp. 1035-1045

13 *Wani, M. A.; Batchelor, B.G.*: Edge-Region-Based Segmentation of Range Images. IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (1994) 3, pp. 314-319