Available online at www.sciencedirect.com
ScienceDirect
Computer-Aided Design 39 (2007) 276–283
COMPUTER-AIDED DESIGN
www.elsevier.com/locate/cad

Detection of closed sharp edges in point clouds using normal estimation and graph theory

Kris Demarsin[a,*], Denis Vanderstraeten[b], Tim Volodine[a], Dirk Roose[a]

[a] *Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, 3001 Heverlee, Belgium*
[b] *Metris N.V., Interleuvenlaan 86, B-3001 Leuven, Belgium*

Received 18 May 2006; accepted 17 December 2006

Abstract

The reconstruction of a surface model from a point cloud is an important task in the reverse engineering of industrial parts. We aim at constructing a *curve network* on the point cloud that will define the border of the various surface patches. In this paper, we present an algorithm to extract *closed* sharp feature lines, which is necessary to create such a closed curve network. We use a first order segmentation to extract candidate feature points and process them as a graph to recover the sharp feature lines. To this end, a minimum spanning tree is constructed and afterwards a reconnection procedure closes the lines. The algorithm is fast and gives good results for real-world point sets from industrial applications.

Keywords: Reverse engineering; Segmentation; Region growing; Sharp edges; Point clouds; Curve network

1. Introduction

Feature lines on a surface model can be mathematically defined via local extrema of principal curvatures along corresponding principal directions. These feature lines can be used in a variety of applications, e.g. visualization, improvement of the mesh quality, shape recognition, quality control and reverse engineering.

Most feature line extraction algorithms rely on a triangular mesh as input, e.g. [1–5]. Few algorithms only use a point cloud, e.g. [6,7]. These existing methods usually result in pieces of unconnected feature lines, making it hard to segment a point cloud or mesh into surface patches, based on these lines.

A method to find the different surface patches with corresponding boundaries is presented in [8]. This method relies on a triangular mesh as input, but sometimes a mesh is not given or it can be hard to generate one. A segmentation algorithm that operates on point clouds is explained in [9]. This method uses curvature information that is difficult to estimate in a noisy environment.

In this paper, we aim at constructing a curve network on a point cloud that will define the border of the various surface patches. Therefore, we focus on finding closed sharp feature lines. Our method can be sketched as follows. First, a region growing method is applied with normal estimation, which is a modification of the method of Vanco et al. [10–12], to cluster the points in order to reduce the point cloud size. We build and manipulate a graph of these clusters, resulting in closed sharp feature lines that fit the clusters such that the algorithm can be used as a pre-process step to find the areas where a surface patch can be defined. We are interested in point clouds from industrial applications, where closed sharp feature lines can be detected. It is not our goal to apply the algorithm to point clouds with free form surfaces or fillets with a large radius.

The algorithm differs from existing feature line algorithms by the fact that it reconstructs *closed sharp* feature lines. The advantages of the algorithm are that (a) it is meshless, i.e. only the coordinates of the points are used, (b) it intelligently clusters the points to create a graph that is much smaller than the original cloud, thus making it practical for large point clouds and, (c) it constitutes a pre-process step for surface reconstruction.

An introduction to the algorithm, together with some results, is published in [13]. In this paper, the algorithm is explained in detail (see Section 2). In Section 3 we illustrate results of the algorithm applied to realistic point clouds, i.e. point

* Corresponding author.
E-mail address: kris.demarsin@cs.kuleuven.be (K. Demarsin).

0010-4485/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2006.12.005

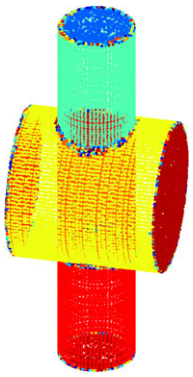

Fig. 1. First order segmentation of two intersecting cylinders.

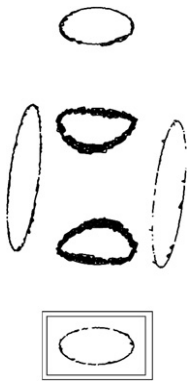

Fig. 2. The graph G_{all}; the area bounded by the rectangle is used to illustrate the following steps of the algorithm in detail.

clouds obtained from scanning industrial parts. We formulate the conclusions in Section 4.

2. Sharp feature line extraction algorithm

2.1. Algorithm overview

Given a point cloud, we extract closed polygonal lines indicating the sharp edges. Algorithm 1 gives the different steps of the algorithm that will be explained in this section. Each step of the algorithm is illustrated in Figs. 1–3 for a point cloud representing two intersecting cylinders.

2.2. First order segmentation of point cloud

Normal estimation and neighborhood selection. The first step of the algorithm divides a point cloud in different clusters of points using a region growing method. For that purpose, we need for every point the neighboring points as well as an estimation for the normal vector. In order to accurately detect the transition from a smooth area to a sharp edge, the normal vector is estimated as locally as possible. As we will see later on, to detect the sharp edges, the neighbors of a point p need to be distributed around p.

With uniform sampling density, we could use a few nearest neighbors to represent the neighborhood of p, but in most realistic point clouds this results in a neighborhood located on one side, which is undesirable in our algorithm. If a triangular mesh is available, the 1-ring neighborhood of p can be used to represent a good local neighborhood of p, with the neighbors distributed around p. In the absence of a mesh, we approximate this neighborhood by building a local mesh. We determine the k nearest neighbors of p, with k large enough, by generating a sphere with p as midpoint and a radius such that the k nearest neighbors are inside this sphere. Then the least squares plane through these points is constructed and the points are projected on this plane. Next, we compute the Delaunay triangulation of these projected points and only the points that share an edge with p in this triangulation, constitute the Delaunay neighborhood [14] of p, which is used as an approximation of the 1-ring neighborhood.

Algorithm 1 High level description of the algorithm.

1. Segment point cloud using the normals $\Rightarrow$ point clusters (*clusters*) (Fig. 1)
2. Build graph G_{all} connecting neighboring clusters (Figs. 2 and 3(a))
3. Add edges, indicating a piece of a sharp feature line, to $G_{all} \Rightarrow G_{extended}$ (Fig. 3(b))
4. Build the pruned minimum spanning tree of $G_{extended} \Rightarrow G_{pruned\_mst}$ (Fig. 3(c))
5. Prune short branches in $G_{pruned\_mst} \Rightarrow G_{pruned\_branches}$ (Fig. 3(d))
6. Close the sharp feature lines in $G_{pruned\_branches} \Rightarrow G_{closed}$ (Fig. 3(e))
7. Smooth the sharp feature lines in $G_{closed} \Rightarrow G_{smooth}$ (Fig. 3(f))

The normal in p is estimated as the normal of the least squares plane through the 1-ring neighborhood of p, as explained in [15]. Since we only use a few points from the k nearest neighbors (the 1-ring neighborhood), the value of k has not much influence on the normal estimation when k is chosen large enough. Consequently, setting the value of this parameter to 20 was suitable for all the tested point clouds.

Region growing. We want to use a standard region growing method to cluster the points based on the sharp edges using only normal estimation. Hence, we use a modification of the first order segmentation described by Vanco et al. [10–12]: only one threshold angle α_{max} is used which specifies the maximum acceptable angle between the normals of two adjacent points in one cluster. At a sharp edge, the normal estimation depends heavily on the computed 1-ring neighborhood, since this neighborhood is very local and these neighbors are located on both sides of the sharp edge. This means that the variation of the normals along a sharp edge is high, resulting in large clusters with low variation of the normals bounded by small clusters with high normal variation. These small clusters indicate the sharp edges. Fig. 1 illustrates the result of the first order segmentation, applied to the point cloud of the two intersecting cylinders, with each point colored corresponding to the cluster it belongs to. Contrary to the method of Vanco et al. [10–12], each cylindrical piece, bounded by sharp edges, consists of only one large cluster with many small clusters defining the boundary.

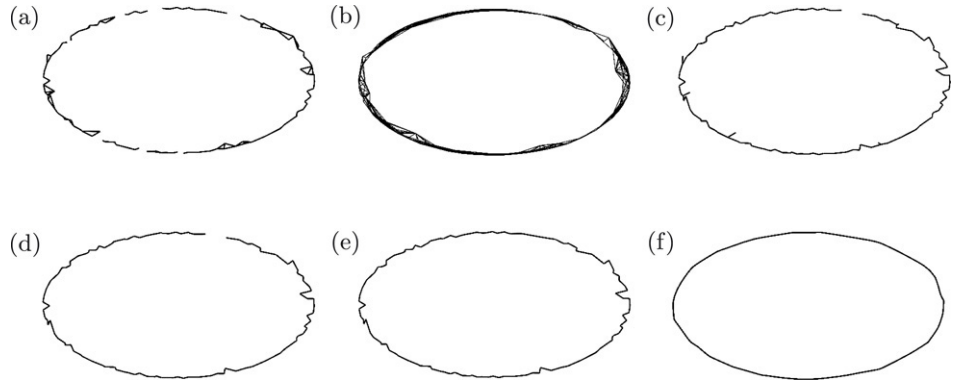

Fig. 3. Result of each step of the algorithm illustrated with the detail of G_{all} indicated by the rectangle in Fig. 2. (a) G_{all}: graph of the clusters; (b) $G_{extended}$: G_{all} with extra edges; (c) G_{pruned_mst}: graph after building the pruned minimum spanning tree of $G_{extended}$; (d) $G_{pruned\_branches}$: graph after pruning short branches in G_{pruned_mst}; (e) G_{closed}: graph after closing the sharp feature lines in $G_{pruned\_branches}$; (f) G_{smooth}: graph after smoothing G_{closed}.

Perfectly aligned points. Since high normal variation is caused by differences in the 1-ring neighborhood, we can wonder how the segmentation behaves on point clouds where the points are perfectly aligned. Even in such an unrealistic case, there are many small clusters at the sharp edges.

2.3. Building graph G_{all} of all clusters

Gumhold et al. [6] and Pauly et al. [7] introduced a graph approach to extract feature lines at the level of individual points. Since we are interested in sharp edges and the segmentation gives a strong indication of the location of these sharp edges, we base the algorithm on a graph structure *at the level of clusters*, which yields a reduction in complexity. Contrary to [6,7], our prime focus is to detect closed sharp feature lines. A connected graph G_{all} is constructed, where each vertex represents a cluster and each edge connects two clusters that contain at least one point with overlapping 1-ring neighborhood. Note that in this paper, an *edge* is a connection between two vertices in a graph and a *sharp edge* is a feature of the point cloud with very high normal variation. For every vertex of the graph we keep a representative point of the cluster, i.e. its center of mass, and its size in terms of points. In Figs. 2 and 3(a) only the edges between two small clusters are plotted. It can be observed that it is these edges that give us a first idea of the location of the sharp feature lines. From now on, we only process the graph and the point cloud is not needed anymore.

Since we have to distinguish between small and large clusters, we could use a user-defined threshold. However, in practice, there are many small clusters and less large clusters with the large clusters much larger than the small clusters and thus taking the average of all sizes of clusters is a good heuristic value to separate the small clusters from the larger ones.

2.4. Adding edges to the graph G_{all} yielding $G_{extended}$

As already mentioned, our goal is to extract closed sharp feature lines, but the graph G_{all} has already many unwanted 'gaps' between small clusters. These gaps are caused by points on a sharp edge with a normal aligned with the normal of a neighboring large cluster. The edges that close these gaps are part of the border between two large clusters, i.e. these edges are located on a sharp edge and have to be included in the graph. Therefore, we add edges to G_{all} connecting two small clusters that share two large neighboring clusters.

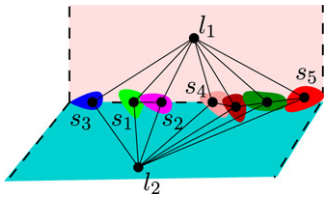

Fig. 4. The graph G_{all} plotted on the segmentation of two perpendicular planes.

This is illustrated in Fig. 4: due to the segmentation a small cluster s_1 may exist that has only one small neighboring cluster, namely s_2, and vice versa, which results in two unwanted gaps. We add the edge (s_3, s_1) to G_{all} since s_3 and s_1 have two common large neighboring clusters, namely l_1 and l_2. Similarly, we add the edge (s_2, s_4) resulting in a graph $G_{extended}$ with no unwanted gaps. We can now understand why it was useful to keep the edges involving a large cluster in the graph G_{all}: the large neighboring clusters are easily found. To avoid the addition of too many edges, e.g. in Fig. 4 we do not want s_1 to get connected with s_5, we only add edges with a euclidean distance between the two representative points which is less than the maximum euclidean distance between two small neighboring clusters in G_{all}. A detail of the graph $G_{extended}$ of the two intersecting cylinders is illustrated in Fig. 3(b). As with G_{all}, only the edges involving two small clusters are drawn.

2.5. Building G_{pruned_mst}, the pruned minimum spanning tree of $G_{extended}$

The edges of $G_{extended}$ involving two small clusters give us an idea of the location of the sharp feature lines. There are many cycles and therefore, we construct the minimum spanning tree (MST) of $G_{extended}$ to remove them. For this purpose, weights of edges between small clusters are calculated as the distance between the representative points of the clusters. Additionally, we attach large weights (larger than the weights between small

clusters) to edges involving a large cluster. Building the MST of $G_{extended}$ with these weights results in a graph with a reduced number of edges of which only a limited number of edges involve a large cluster. We can now remove these latter edges which results in G_{pruned_mst}, as illustrated in Fig. 3(c).

2.6. Construction of $G_{pruned\_branches}$ by pruning short branches in G_{pruned_mst}

Although constructing the pruned minimum spanning tree gives an initial reconstruction of the feature lines, the graph G_{pruned_mst} contains many short branches, as can be seen in Fig. 3(c). It is desirable to remove such branches since they do not correspond to actual features.

We use Algorithm 2 to prune using a maximum branch length d_{max}. This is a similar method as in [6]. If all the branches starting in a point are shorter or longer than the parameter d_{max} then nothing is pruned in this point. In most realistic cases, the noisy branches are shorter than the other ones and, consequently, a good value for d_{max} can be found such that the noisy branches are pruned. For the tested point clouds in this paper, no input of the user was needed to fine-tune d_{max}. Setting the value of this parameter to 5 gave good results for all the tested models. A detail of the resulting graph $G_{pruned\_branches}$ for the two intersecting cylinders can be seen in Fig. 3(d).

Algorithm 2 Pruning algorithm.

$d_{max} = 5$
for all points p in the graph G_{pruned_mst} with *more than* 2 incident edges **do**
  if minimum 2 incident branches have depth $\geq d_{max}$ **then**
    remove all incident branches with depth $< d_{max}$
  end if
  if (exactly 1 incident branch has depth $\geq d_{max}$) **and** (minimum 2 incident branches have depth $< d_{max}$) **then**
    from the short branches, keep only the one with the largest depth
  end if
end for

2.7. Building G_{closed} by closing the sharp feature lines in $G_{pruned\_branches}$

At this point, we have a graph $G_{pruned\_branches}$ with a reduced number of endpoints, i.e. vertices in the graph with exactly one incident edge. Since our aim is to reconstruct *closed* lines, we introduce a 'connect' algorithm to link each endpoint with a suited point in the graph as follows: for each endpoint p_i we compute the set N of the n nearest neighbors:

- *Case* 1: there exists at least one other endpoint in N:
  The endpoint p_i is connected to the closest endpoint q_i in N which has no path with p_i that is too short, i.e. the number of edges of each path is above a certain threshold d. In this way, we avoid generating small cycles, e.g. in Fig. 5 there exists a path between p_1 and q_2 which is too short and, consequently, we do not connect them. A good endpoint to link p_1 with is q_1: they have only paths which are long enough. If no suitable endpoint is found in N, we do not link p_i to anything, i.e. it stays an endpoint, like endpoint p_2 in Fig. 5.

  Since we have to search *all* the paths between p_i and q_i and because there can be many (long) paths, we optimize the algorithm by performing a depth first search with a restricted depth d, i.e. if at depth d there is still no path found between p_i and q_i, there are two possibilities: there is a path which is long enough (longer than d) or there is no path. In both cases, it is unnecessary to go deeper in the search tree, since no small cycles will be generated following the current path and we continue searching the other paths.
- *Case* 2: there exists no other endpoint in N:
  This case is similar to the previous case, but now we link p_i with a point in N which has more than one incident edge. For example p_4 is linked to q_4, but p_3 stays an endpoint, since all points in N would generate a small cycle when connected with p_3.

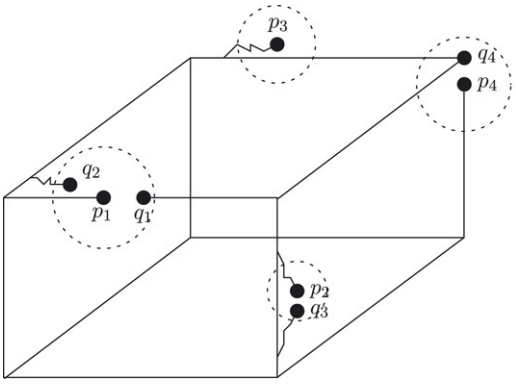

Fig. 5. Illustration of the connect algorithm. The circles with p_i as midpoint for $i \in \{1, 2, 3, 4\}$ indicate the spheres containing the n nearest neighbors of p_i.

If, after the connect algorithm, there still exist endpoints, we prune all remaining branches, e.g. in Fig. 5 the four short branches ending in respectively p_2, q_3, q_2 and p_3 will be pruned. The connect algorithm can be seen as a last clean up step: 'noisy' branches, which are restricted to a minimum because of the previous steps of the algorithm, are pruned and all sharp feature lines are closed.

The value of n must be chosen carefully to ensure that the needed points to link with are in N and to avoid connections with points that are located too far away. The value of d must be chosen such that all the sharp feature lines are extracted and all noisy branches are removed. The graph $G_{pruned\_branches}$ gives an indication of the length of the closed loops we want to reconstruct, which can aid in a first estimate for d. By studying the neighboring points of the endpoints, the user gets an initial estimate for n. Both parameters can be adjusted by the user until the desired result is obtained.

The two endpoints that still existed in Fig. 3(d) are now connected, as illustrated in Fig. 3(e).

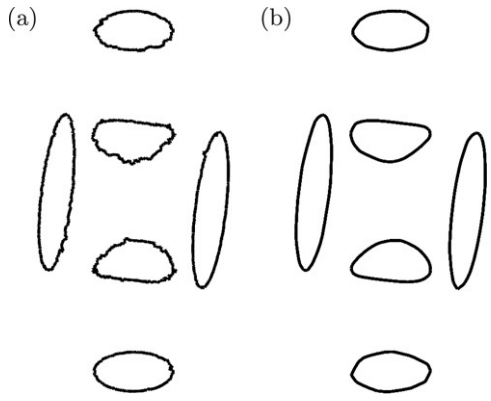

Fig. 6. Results for the two intersecting cylinders. (a) G_{closed}; (b) G_{smooth}.

2.8. Generating G_{smooth} by smoothing G_{closed}

The method we use to get a smooth graph G_{smooth}, see Fig. 3(f), is explained in [16].

3. Results

A detail of the two intersecting cylinders was used to illustrate the different steps of the algorithm. The whole graphs G_{closed} and G_{smooth} are shown in Fig. 6(a) and (b). The results for a cube can be seen in Fig. 7: we see how the connect algorithm results in perfectly closed sharp feature lines. Fig. 8 shows the results for a detail of a mobile phone, a typical example of a point cloud used in industrial applications. This point cloud has been generated by a Metris LC15 scanner.[1] We note that the algorithm does not guarantee good results when two sharp feature lines are located too close to each other depending on neighborhood selection and point density: there is an unwanted gap at the right caused by the pruning, and there exist some edges connecting two sharp feature lines. One way to partially solve the latter problem would be to generate an ellipsoid in the direction of the sharp feature line instead of a sphere to find the neighboring points to connect to in the close step.

The results for a larger part of the mobile phone are illustrated in Fig. 9. Contrary to the previous point clouds, this point cloud does not represent a solid, i.e. it has a boundary which we extract and include in the graph. By comparing the final graph with the segmentation, we see how well the detected lines fit the segmentation, but we also see that a few sharp feature lines are not detected because they consist of cycles which are too short. Additionally, where two lines are located too close to each other, they are extracted as one line.

A final point cloud is illustrated in Fig. 10. Five cylinders are pointing out from a plane, with the small one pointing in the opposite direction to the four other cylinders. This plane is bounded by sharp edges caused by four neighboring planes orthogonal to this plane. At the other side, these neighboring planes constitute the boundary of the point cloud. This cloud has an average distance between a point and the corresponding 1-ring neighbors of 1/2 mm and the noise is estimated as 30 μm (single sigma of local least squares plane). The noisy parts of the point cloud produce many noisy branches in $G_{pruned\_branches}$, especially in the neighborhood of the four cylinders (see Fig. 10(c)). These branches are removed in the final graph G_{smooth}, illustrating the importance of the connect algorithm in the presence of noise. The small cylinder is entirely reconstructed, since it is closed at the top. Only the bottom circles of the other four cylinders are reconstructed, since these cylinders are open on the other side, i.e. they constitute four holes in the point cloud. Since the boundary is not extracted for this cloud, connecting edges between the sharp edges and the boundary are missing in the final graph, i.e. they are removed from $G_{pruned\_branches}$ by the connect algorithm.

Table 1
Information about the segmentation for the different point clouds

	#Points	Avg	#Small clusters	#Large clusters
Phone small	11 034	1.24	1610	37
Cube	15 206	1.17	851	6
Cylinders	26 846	1.11	1696	10
Brick	41 432	1.43	3133	36
Phone large	110 053	1.62	5247	69

The column Avg shows the average number of points of the small clusters for each data set.

This latter point cloud, the brick example, illustrates how the algorithm is influenced by noise: small clusters are created at the noisy parts. When we apply the algorithm to point clouds with very high noise, the segmentation step results in small clusters without any large ones. In such cases, a smoothing method should be applied first and a larger neighborhood for normal estimation can be used. In this paper, it is not our intention to give a deep analysis of the algorithm in the presence of high noise.

Table 1 presents information about the segmentation. Because of the high normal variation at the sharp edges, the average size of a small cluster is close to unity. In the case of the cube and the cylinders, we see that the segmentation results in the correct number of large clusters: the large clusters fit the extracted sharp feature lines perfectly. Table 2 gives for every step of the algorithm the number of vertices and edges of the corresponding graph. In the case of the two intersecting cylinders, we start with a point cloud of 26 846 points and then we build a graph G_{all} of 1706 vertices and 6086 edges. After adding edges to G_{all}, every following step reduces the memory consumption of the graph: a huge reduction in the number of edges happens when building G_{pruned_mst} and $G_{pruned\_branches}$. In general, in the close step, more edges are removed than added, since noisy branches are pruned. Note that for the large mobile phone point cloud the boundary is included just before the close step.

The table also illustrates the time consumption of the algorithm. The segmentation step ($\vec{n}$ + RG) requires more time compared to the other steps of the algorithm, because this step has to grow through all the points of the point cloud and the normal for each point needs to be estimated. We

[1] www.metris.com.

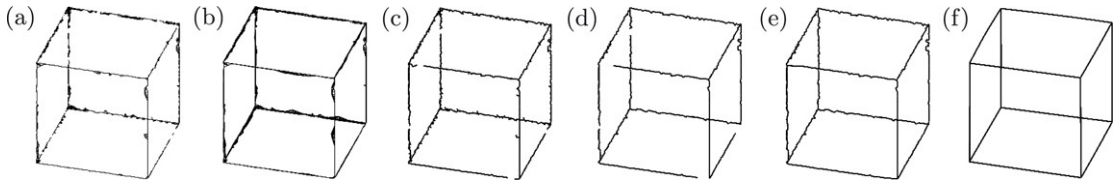

Fig. 7. Result of each step of the algorithm applied to the cube. (a) G_{all}; (b) $G_{extended}$; (c) G_{pruned_mst}; (d) $G_{pruned\_branches}$; (e) G_{closed}; (f) G_{smooth}.

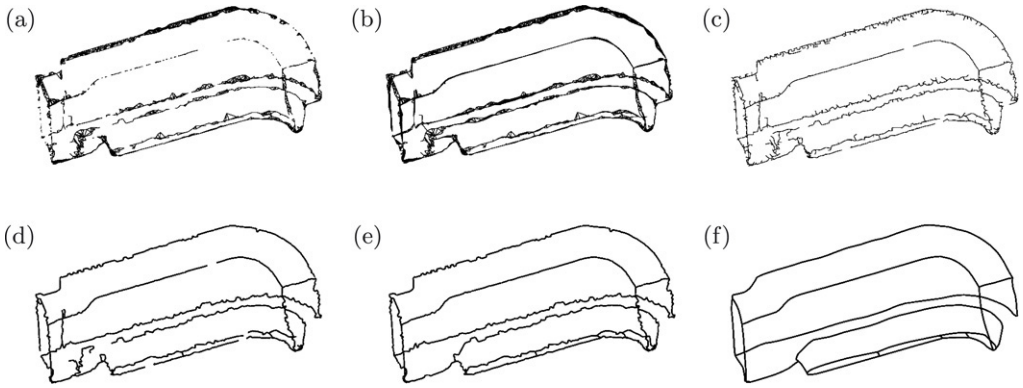

Fig. 8. Result of each step of the algorithm applied to the mobile phone point cloud. (a) G_{all}; (b) $G_{extended}$; (c) G_{pruned_mst}; (d) $G_{pruned\_branches}$; (e) G_{closed}; (f) G_{smooth}.

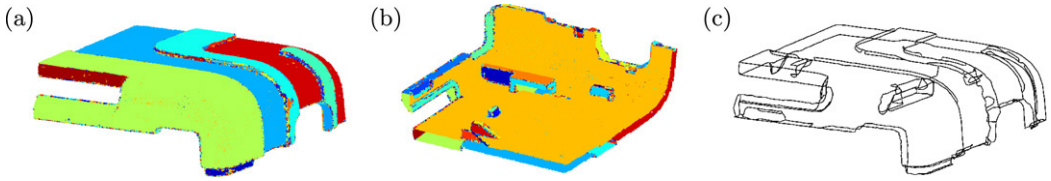

Fig. 9. Results for a larger detail of the mobile phone. (a) and (b) illustrate two different views of the segmentation; (c) G_{smooth}.

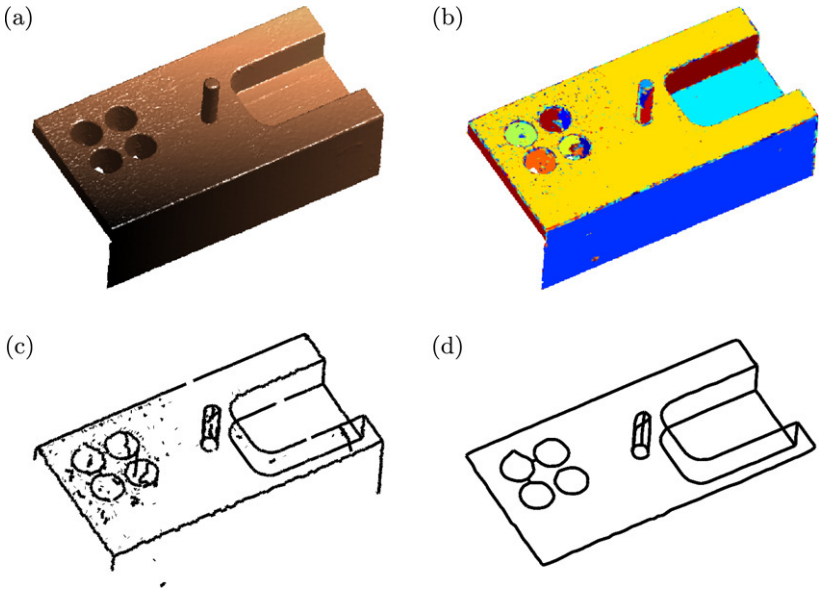

Fig. 10. Results for the brick point cloud. (a) Point cloud with flat shading; (b) segmentation; (c) $G_{pruned\_branches}$; (d) G_{smooth}.

Table 2
Complexity and time consumption of the different steps

Edges	G_{all}	$G_{extended}$	G_{pruned_mst}	$G_{pruned\_branches}$	G_{closed}
Phone small	5206	8776	1604	1225	1170
Cube	2835	5251	850	685	690
Cylinders	6086	8984	1690	1123	1076
Brick	8210	9716	2543	1992	1456
Phone large	16 198	27 538	5199	4040	4579
Vertices					
Phone small	1647	1647	1607	1228	1162
Cube	857	857	851	686	686
Cylinders	1706	1706	1696	1129	1076
Brick	3169	3169	2720	2169	1449
Phone large	5316	5316	5215	4056	4527

Time (s)	$\vec{n}$	RG	G_{all}	$G_{extended}$	G_{pruned_mst}	$G_{pruned\_branches}$	G_{closed}	Total
Phone small	1.69	0.11	0.18	0.39	0.05	0.04	0.09	2.55
Cube	2.02	0.11	0.08	0.25	0.03	0.01	0.02	2.52
Cylinders	3.71	0.20	0.23	0.49	0.06	0.04	0.08	4.81
Brick	6.06	0.58	0.33	1.43	0.08	0.08	0.71	9.27
Phone large	16.01	1.06	0.66	2.32	0.17	0.12	0.96	21.30

The segmentation is decomposed in two steps: The normal estimation ($\vec{n}$) and the region growing (RG). The timings are in seconds and generated on an Intel Pentium 4, 3.20 GHz.

could make the segmentation much faster by estimating the normal as the normal of the least squares plane through the k nearest neighbors instead of using the 1-ring neighborhood (see Section 2.2). But for realistic point clouds, a small k might generate neighbors only on one side of the sharp edge causing no small clusters, and a large k makes it impossible to accurately detect the transition from a smooth surface to a sharp edge.

The following parameters are used by the algorithm: α_{max} (angle in the segmentation), k (to estimate the normal), d_{max} (needed in the pruning algorithm), and n and d (used in the connect algorithm). Most of these parameters have the same value for all tested models: $\alpha_{max} = 8°$, $k = 20$ and $d_{max} = 5$. Only a few parameters need to be specified by the user, namely n and d in the connect algorithm (see Section 2.7).

4. Conclusion and future work

We presented an algorithm to extract sharp edges from a point cloud without estimating the curvature and without triangulating the point cloud. Additionally, all extracted lines are *closed* at the end of the algorithm. We start with a very simple region growing method with well chosen normals, resulting in an initial clustering based on the sharp edges. Afterwards, we build and manipulate a graph of the clusters. Using a graph structure at the level of clusters yields faster execution times and less memory consumption, making the algorithm suitable for large point clouds. Once we build the graph of the clusters, the point cloud is no longer needed and we only process the graph in the following steps: adding extra edges, construction of the minimum spanning tree, pruning, closing and smoothing the sharp feature lines.

The segmentation step and the creation of the *closed* lines constitute a pre-process step in finding a curve network. In the future, we plan to construct this network, which consists of a set of loops, where each loop defines the boundary of an area where a patch can be fitted. When all these segments are known, we can continue with each segment individually to detect also tangent continuous but curvature discontinuous features like fillets.

Acknowledgement

The two mobile phone point clouds and the brick cloud are courtesy of Metris N.V. Belgium.

References

[1] Hildebrandt K, Polthier K, Wardetzky M. Smooth feature lines on surface meshes. In: Symposium on geometry processing. 2005. p. 85–90.
[2] Ohtake Y, Belyaev A. Automatic detection of geodesic ridges and ravines on polygonal surfaces. The Journal of Three Dimensional Images 2001; 15(1):127–32.
[3] Ohtake Y, Belyaev A, Seidel HP. Ridge-valley lines on meshes via implicit surface fitting. In: SIGGRAPH. 2004. p. 609–12.
[4] Stylianou G, Farin G. Crest lines extraction from 3D triangulated meshes. In: Hierarchical and geometrical methods in scientific visualization. 2003. p. 269–81.
[5] Watanabe K, Belyaev AG. Detection of salient curvature features on polygonal surfaces. Computer Graphics Forum 2001;20(3):385–92.
[6] Gumhold S, Wang X, MacLeod R. Feature extraction from point clouds. In: Proceedings of the 10th international meshing roundtable. 2001. p. 293–305.
[7] Pauly M, Keiser R, Gross MH. Multi-scale feature extraction on point-sampled surfaces. Computer Graphics Forum 2003;22(3):281–90.
[8] Meyer A, Marin P. Segmentation of 3D triangulated data points using edges constructed with a C1 discontinuous surface fitting. Computer Aided Design 2004;36:1327–36.
[9] Yang M, Lee E. Segmentation of measured point data using a parametric quadric surface approximation. Computer Aided Design 1999;31:449–57.

[10] Vanco M, Brunnett G, Schreiber Th. A direct approach towards automatic surface segmentation of unorganized 3d points. In: Proceedings spring conference on computer graphics. 2000. p. 185–94.
[11] Vanco M, Brunnett G. Direct segmentation for reverse engineering. In: Proceedings international symposium on cyber worlds. 2002. p. 24–37.
[12] Vanco M, Brunnett G. Direct segmentation of algebraic models for reverse engineering. Computing 2004;72(1–2):207–20.
[13] Demarsin K, Vanderstraeten D, Volodine T, Roose D. Detection of closed sharp feature lines in point clouds for reverse engineering applications. In: Proceedings of geometric modeling and processing. 2006. p. 571–7.
[14] Floater MS, Reimers M. Meshless parameterization and surface reconstruction. Computer Aided Geometric Design 2001;18(2):77–92.
[15] Hormann K. Theory and applications of parameterizing triangulations. Ph.D. thesis. Department of computer science, University of Erlangen; 2001.
[16] Volodine T, Vanderstraeten D, Roose D. Smoothing of meshes and point clouds using weighted geometry-aware bases. Report TW 451. Belgium: Department of computer science, K.U. Leuven; 2006.