Crest Line Extraction From Point Clouds

Georgios Stylianou
Department of Computer Science
Cyprus College
gstylianou@cycollege.ac.cy

Yiorgos Chrysanthou
Department of Computer Science
University of Cyprus
yiorgos@ucy.ac.cy

Abstract

We present a simple, automatic method for extracting feature curves, called crest lines, from point clouds. Crest lines are surface shape features having a mathematical background. Given an unstructured point cloud as input, we pre-process the data to generate some topological information by creating an undirectional surface graph. The method starts with the approximation of normal curvature on every point. Utilizing the crest point definition, the crest points are identified. Region growth follows to implicitly connect the crest points and create crest graphs. Finally, the computation of minimum spanning trees for every crest graph and the pruning of short branches create crest lines.

Keywords: crest lines, point clouds, principal curvatures.

1 INTRODUCTION

Modern acquisition techniques, i.e. laser scanners, provide us with a great amount of data sets. Their major characteristic is the ability to produce sample points in the magnitude of millions. As a consequence, point-sampled surfaces have emerged in the last years, as an alternative representation to triangulated surfaces, especially for the purpose of rendering. The surface of a 3D object is represented by a set of sample points without any other topological information. To display such models, various point rendering systems have been developed [Pfister et al. 2000; Rusinkiewitz and Levoy 2000; Zwicker et al. 2001; Alexa et al. 2001; Fleishman et al. 2003]. In addition, processing applications such as simplification [Pauly et al. 2002a] and editing [Zwicker et al. 2002] have been introduced.

In this paper, we introduce a simple, automatic method for extracting shape features, called crest lines, from point clouds. These have been used in various applications for over a decade. Crest lines are useful in many fields because they identify vital features of virtually any surface. Their mathematical definition is general enough to include applications from different areas. They provide us a satisfactory geometrical representation of important physical properties such as ridge lines and valleys in the case of aerial images [Monga et al. 1997], or anatomical features in the case of medical images [Khaneja et al. 1998; Thirion and Gourdon 1996]. They have been used, among others, for object registration [Guéziec and Ayache 1992; Stylianou 2004], growth simulation [Andresen et al. 1998], automatic retrieval of anatomical structures [Declerck et al. 1995].

2 Related Work

There are several methods proposed for extraction of crest lines or otherwise defined features from surfaces. Related work regarding crest lines includes Lengagne et al. [Lengagne et al. 1996] who define crest points as the zero crossings of the directional derivative of the largest curvature in its direction along the edges of a triangulated mesh, Guéziec [Guéziec 1993] extracts crest lines from B-spline surfaces, Khaneja et al. [Khaneja et al. 1998] extract geodesic lines on triangulated surfaces using user-specified start-end points that under certain constraints are crest lines. Thirion et al. [Thirion and Gourdon 1996] extract crest lines as the intersection of two surfaces. Ohtake and Belyaev [Ohtake and Belyaev 2001] compute ridges and ravines using local differential geometry operators. Also, Ohtake et al. [Ohtake et al. 2004] compute ridges and valleys by first fitting an implicit polynomial to the surface. Hubeli and Gross [Hubeli and Gross 2001] used a multiresolution approach for feature extraction on triangle meshes. They developed various classification operators used to identify a set of feature edges. Stylianou [Stylianou 2003] has described an automatic method for extracting crest lines from triangulated meshes. He used finite differences for crest point classification and dilation, erosion for creating crest lines.

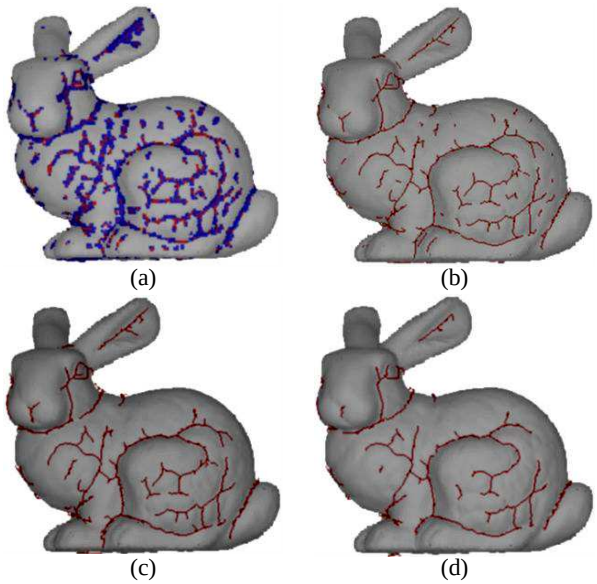


Figure 1: Method pipeline: extracting valley features. (a) Crest point classification and region growing, (b) skinning, (c) removal of small branches and (d) removal of small lines.

The most notable work involved with feature extraction from point clouds are Pauly et al. [Pauly et al. 2003] and Gumhold et al. [Gumhold et al. 2001]. Gumhold et al.'s method starts by creating the neighbor graph that goes as far as triangulating the surface up to a certain degree. Covariance analysis is used for feature classification. This method categorizes the features into five categories and computes a separate penalty weight for each feature. The results are not so good because even though they detect features correctly they cannot detect all the features. A possible reason is that the undetected features do not fall in any of the defined feature categories.

Pauly et al. describe a multiscale approach towards feature extraction where the main component is principal component analysis for measuring surface variation and the assignment of confidence weights. Their method performs quite well but has the disandvan-

tage of being semi-automatic; the user has to specify some thresholding parameters. Also, their implementation can suffer from incosistencies on the k-closest neighbor computation due to sampling, as it is clarified later.

The contribution of this work is:

- Simple. We propose a method that is easy to implement and use because it does not make use of any arbitrary weights and thresholds.
- Automatic. The method does not require user intervention to successfully extract the surface features. User intervenes, if desires after the termination of the algorithm, only to interactively eliminate features to keep the most important.
- Complete. It successfully identifies all of the surface's features. This is shown experimentally.
- General. The method is general as it does not make any assumptions regarding the type and geometry of the surfaces.

3 Crest Line Definition

A parametric surface is a mapping from $\Re^2$ to $\Re^3$, $\mathbf{x}(\mathbf{u}) = (x(u,v), y(u,v), z(u,v))^T \subset \Re^3$, $\mathbf{u} = (u,v) \subset \Re^2$ is the domain. Normal curvature measures the bending of the surface locally on every point $\mathbf{x}(\mathbf{u})$. Because there are infinitely many directions to compute normal curvature, the standard approach is to compute only the largest and smallest (principal) curvatures denoted by k_1, k_2, respectively. These have associated directions $\mathbf{t}_1$, $\mathbf{t}_2$ which are orthogonal. Principal curvatures can be positive or negative, with the sign denoting whether the surface bends outwards or inwards. Largest curvature k_1 is larger than curvature k_2 in absolute value ($|k_1| > |k_2|$). A crest point is a point of the surface where its largest curvature k_1 is maximum in its corresponding direction. The definition of a crest point is

$$D_{\mathbf{t}_1} k_1(u,v) = 0 \tag{1}$$

where $D_{\mathbf{t}_1}$ is the directional derivative in direction $\mathbf{t}_1 \in \Re^2$, $k_1 \in \Re$ is the largest principal curvature and $\mathbf{t}_1$ is its domain direction on a point (of the surface) with domain coordinates (u,v).

Crest points implicitly trace lines on the surface denoted as crest lines. Crest lines are shape features with the main characteristic of using local information to yield a global description of the surface. Even though crest lines are local shape features, when they are viewed together they partially describe the surface.

Crest points trace ridges (when k_1 is positive) and valleys (when k_1 is negative) on the surface. Even though we can trace ridges and valleys concurrently, it is as useful to trace only ridges or valleys, especially because ridges have different characteristics than valleys, like their k_1 curvature ranges, even though they are features of the same surface.

Figure 2 shows an example of a crest line. Because a crest point has maximum largest curvature in its corresponding direction, a crest line naturally follows the direction of the smallest curvature of its composing crest points.

4 OVERVIEW

The input is a surface represented by an unstructured point cloud $P = \{\mathbf{p}_i\} \in \Re^3$. This introduces several "problems" in applying the crest point definition directly. Normal curvature estimation cannot directly be achieved because of the lack of structure. Therefore we preprocess the data, with aim to add some topological information,

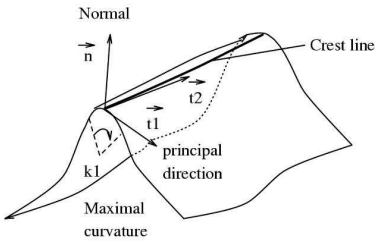


Figure 2: A crest line example.

by transforming the point cloud into an undirectional graph. In addition, our method assumes that on every point we know *a priori* its normal vector. This is not a problem because there are several techniques in the literature about normal vector estimation [Hoppe et al. 1992; Pauly et al. 2002b]. Furthermore, laser scanners that provide us with several data sets, provide the normal vectors, as well.

After, topological information is created for the point cloud, the method for calculating crest lines proceeds as follows:

1. Approximate normal curvature on every point of the surface.
2. Identify all crest points.
3. Apply region growing around crest points to create crest graphs.
4. Compute a minimum spanning tree for every crest graph.
5. Prune short branches.
6. Remove small crest lines, to eliminate artifacts or small features (optional).

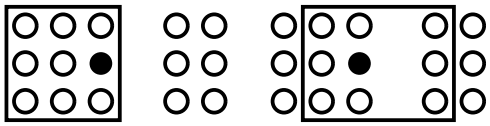
Figure 1 shows an example of the method pipeline. First, we identify crest points (red points) and perform region growth (blue points). Second, we compute the minimum spanning tree of all remaining graphs. Third, we prune short branches. Finally, we remove small lines. This is interactive and the size of the smallest line to remain is the user's decision. Usually, small features are considered artifacts.

4.1 Generating topological information

We impose structure by converting the point cloud to an undirectional graph. We use the k-closest neighbors operator [Pauly et al. 2003; Gumhold et al. 2001] to create the local neighborhood graph for every point $\mathbf{p}$; this connects $\mathbf{p}$ with the 1-ring neighboring points. ANN library [Arya et al. 1998] is used to approximate k-closest neighbors in near linear time. This graph cannot be used directly, though, because it suffers from sampling inconsistencies. When the sampling is not uniform with regard to Euclidean distance between samples, problems emerge. Figure 3a shows an example. All or most of the closest neighbors (outlined circles) of point $\mathbf{p}$ (filled circle) are located to one side of $\mathbf{p}$ and not around it as they should. This drawback results in a bad approximation of normal curvature on those points. This is solved by getting more neighbors and filtering them to keep neighbors that are located around the point (figure 3b).

The filtering (figure 4) of the closest neighbors of point $\mathbf{p}$ is:

- Compute plane P passing through point $\mathbf{p}$ and having the normal vector at $\mathbf{p}$ as its normal.
- Define an orthonormal coordinate system on P with $\mathbf{p}$ as its origin and two arbitrary unit vectors $(\mathbf{x}, \mathbf{y})$ in P.



(a) (b)

Figure 3: 2D example. The closest Neighbors are in the rectangle.(a) k-closest neighbor operator and (b) after filtering.

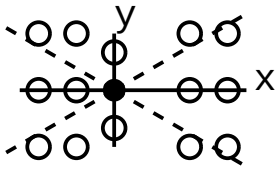


Figure 4: 2D example. Filtering k-closest neighbors. The dashed lines show the intervals.

- Project (using orthogonal projection) all neighbors of point **p** onto plane P and represent their projections with respect to the local coordinate system in P.
- The unit vectors of the local coordinate system divide the plane into four quadrants $(\mathbf{x}^+,\mathbf{y}^+)$, $(\mathbf{x}^-,\mathbf{y}^+)$, $(\mathbf{x}^-,\mathbf{y}^-)$, $(\mathbf{x}^+,\mathbf{y}^-)$.
- Select at least two closest points located in each quadrant. These are the neighbors of **p**.

The closest points of each quadrant are selected using their polar angles, depending on the quarter they are located, and their distance to point **p**. The idea is that the points to be selected must have different angular distance from an axis. If the angular distance of two points is the same, then the more distant point is not really a neighbor and is useless for normal curvature approximation. Therefore, supposing we want to get two points per quadrant we use intervals of 45 degrees; then we get the closest point with angular distance between $(0,45]$ degrees and the closest point between $(45,90]$ degrees. Selection of three points means intervals of 30 degrees and so on. In the case where two or more points equidistant in an interval we select all of them. In this work, we select about two points per quadrant. It is reasonable to select in total eight points because the mean vertex star size in triangulations is six.

Figure 4 shows an overview of this algorithm. This algorithm works well with the assumption that sampling is relatively dense. Otherwise, we would have to go far to find neighbors, violating the prerequisite that all points of the neighborhood must belong to the same connected region of the underlying surface. If it is necessary to go far, a simple heuristic to avoid this problem is to look for jumps in the directions of the normal vectors of the neighbors of point **p**. In addition, when points are on the boundary it is not possible to locate neighbors in all four regions. This can serve as a criterion for boundary point identification. Finally, we filter the neighbors to keep only few neighbors, quite close and useful for curvature approximation.

Even though the neighbor graph, after filtering, is good, it is not consistent. Creating an edge from point $\mathbf{p}_1$ to point $\mathbf{p}_2$ does not necessarily mean that an edge from $\mathbf{p}_2$ to $\mathbf{p}_1$ will exist. This is a natural outcome of the whole process because the closest neighbor operation, that reminds us somewhat the directed Hausdorff distance, is not a metric. We solved this inconsistency by visiting every point **p**, getting all its neighbors $\mathbf{p}_i$ and assigning **p** as $\mathbf{p}_i$'s neighbor, if it was not already a neighbor.

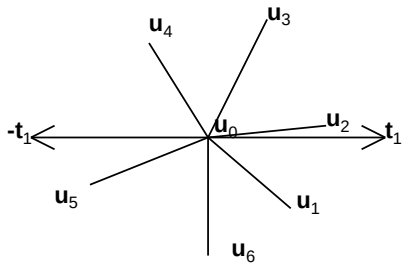


Figure 5: Crest point classification.

4.2 Curvature Approximation

The problem with a point graph is that we cannot explicitly calculate derivatives and curvatures on every point. Therefore, prior to crest point classification, we locally parameterize the surface by fitting a quadric on the neighborhood of every point and calculating its principal curvatures and directions [Farin 2001; Hamann 1993]. The local neighborhood of a point **p** of a point graph contains all the points (star(**p**)) that share an edge with **p** (1-ring points). The local domain values of **p** and star(**p**) are calculated by projecting points **p** and star(**p**) on the plane defined by the normal vector of **p**. The normal vector of **p** is either estimated or comes with the data. Note that the local domain values of **p** and star(**p**), which now are co-planar points, are still a graph.

Note that the local approximations do not guarantee global smoothness or continuity. But this method is proven experimentally that best approximates the principal curvatures. In addition, a whole surface fitting method is not a necessity since a crest point is a local surface feature. Also, it is not easy, if possible, to fit a surface to a point cloud since the exact topology is not really known. Even triangulation algorithms can fail for certain point clouds.

4.3 Crest point Classification

The classification of crest points is achieved using finite differences on the domain of every point **p**. For every point **p** we have already computed its principal curvatures. We are interested only on the largest principal curvature k_1 and its domain direction $\mathbf{t}_1$. We also treat all the domain values of the $star(\mathbf{p})$ $\mathbf{u}_i$ as vectors originating from $\mathbf{u}_0$ (the domain of **p**).

The curvature k_1 should be maximum in its direction $\mathbf{t}_1$. Therefore, suppose $\mathbf{u}_i$ is closer to $\mathbf{t}_1$ and $\mathbf{u}_j$ is closer to $-\mathbf{t}_1$. Then If $(k_1^0)^2-(k_1^i)^2>M$ and $(k_1^0)^2-(k_1^j)^2>M$ then **p** is a crest point. k_1^i is the largest principal curvature of point i. The threshold M controls the level of maximality. It is used to reduce numerical errors and threshold on the number of crest points to keep the most important. A good value is $M=0.01$.

For example using figure 5, we get the largest curvatures on points $\mathbf{u}_2$, $\mathbf{u}_5$. If the largest curvature on vertex $\mathbf{u}_0$ satisfies the criterion, then $\mathbf{u}_0$ is a crest point.

4.4 Region Growing

So far we have identified the crest points. If the surface(s) were really smooth, crest points would implicitly have created all the crest lines. But we do not totally achieve this goal. Noise and irregularities of the point cloud play an important role. The approach towards the solution is to use the crest points identified and perform a region growth around them. Using region growth we implicitly connect all the crest points and create crest regions. For every crest point **p**, we classified as 'crest points by growth' all the points sharing an edge with **p** and are not crest points.

4.5 Minimum Spanning Graph

Every subgraph must be skinned to be left with crest lines. We skin all crest graphs Q by computing the minimum spanning tree (MST) of each and every graph. MST is the minimum cost tree, based on some edge cost, that visits all points of the graph. Prior to the computation of the MST, we have to define a weight for each of the edges of graphs Q. There are three types of weights.

Type A: when an edge is composed of two crest points, the weight is one.

Type B: when an edge is composed of one crest point and a 'crest point by growth' the weight is set to five.

Type C: the rest of the edges have weight ten. These are composed of two 'crest points by growth'.

The weights force the spanning tree to use all type A edges. If necessary it uses type B edges. Type B edges connect type A edges. Type C edges connect type B edges.

The computation of the minimum spanning tree uses the Kruskal algorithm [Cormen et al. 1993]. After the computation of the MST we initially prune all edges of the tree of length one.

4.6 Branch Pruning

This step simplifies the features by iteratively pruning several small branches. Experimentally pruning all branches of size up to ten, produces good results very fast. Pauly et al. [Pauly et al. 2003] suggest to count the size of all possible paths of a each tree (feature) and keep the longest. This is quite slow and there is no quarantee that the longest path is the best path. In addition, it is quite normal for a feature to be composed of various equally important branches.

4.7 Removing Small Lines

At this point the proposed method has terminated. Now, the user if he/she wishes has the opportunity to remove small crest lines; these can be either artifacts or just insignificant features. Crest line removal is interactive and to assist the user a crest line size distribution chart is provided. We name the threshold for maximum line size to remove S.

5 Results

We have implemented the method as described and have tested it on various point sampled models 6. Table 1 summarizes the time complexity that demonstrates the speed of the algorithm. Note that we did not do any optimizations. We have experimented on the Stanford Bunny (35905 points), a statue Igea (33587 points) and a Dinosaur model (28098 points). Since we have point clouds, we used a simple implementation of QSplat [Rusinkiewitz and Levoy 2000], a point cloud renderer, to render the results. On figure 1 we have rendered the outcome of the first image color coded. Rendering crest lines color coded did not produce nice images because every point is rendered as a rectangle. Therefore, even though the feature lines are part of the surface, they are rendered as lines over the point cloud surface.

Pauly et al. [Pauly et al. 2003] and Gumhold et al. [Gumhold et al. 2001] suggested to smooth the feature lines, using B-splines, prior to visualization because this will produce nicer images. Smoothing produces better feature lines, especially for rendering, but these lines are no longer a part of the surface. They are located very closely around the point cloud surface. This can complicate tasks like simplification that can potentially make use of the reconstructed features because the features cannot be used directly.

Table 1: Time Complexity in seconds on a Mobile Intel Pentium III, 1GHz.

Model	Points	Time
Bunny	35905	27
Igea	33587	24
Dinosaur	28098	21

Figure 7 shows the crest lines of the Bunny, Igea and Dinosaur. Red color is used for valleys and blue color for ridges. For the Bunny's valleys, we have set $M = 1$ and $S = 15$. For Bunny's ridges, we have set $M = 2$ and $S = 20$. For Igea's and Dinosaur's ridges and valleys we have set $M = 0.01$ and $S = 10$, which were the defaults. Note, that most of the features, if not all, of both surfaces are identified and they have many important branches. Largest curvature distribution is usually different, depending on the surface, on valleys and ridges giving different good thresholds M. For instance, Bunny's valleys are much more pronounced than its ridges. This explains why different M's are used.

Figure 8 shows how changing threshold M, that controls the level of maximality, affects the results. Sliding M from 0.01 to 6 can delete many features keeping the most important but on the way it can remove other important features like the crest line on the ear.

This method compares very well with Pauly's and Gumhold's because it is more accurate as it extracts shape features using a mathematical formula and not ad hoc methods;thus it can identify easier, without user interaction the features. Also, the implementation itself is easier and direct as we do not make any assumptions and especially the edge weights used for the spanning tree are much simpler and intuitive.

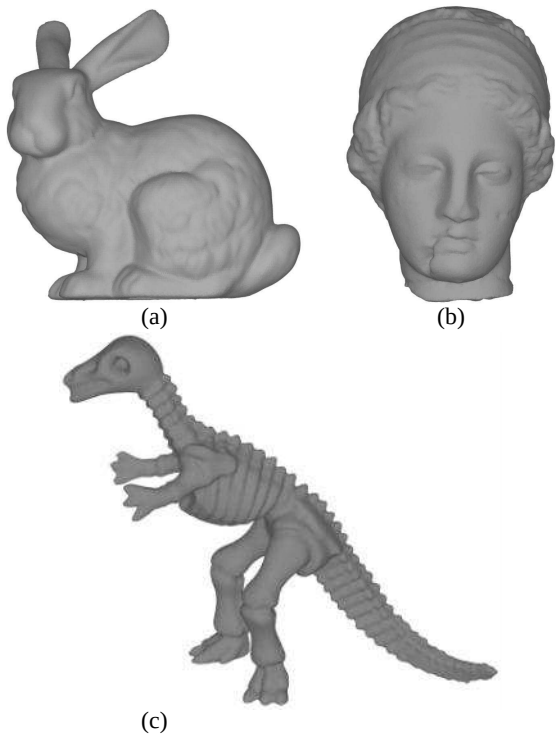


Figure 6: The models. (a) Stanford bunny, (b) Hygea and (c) Dinosaur.

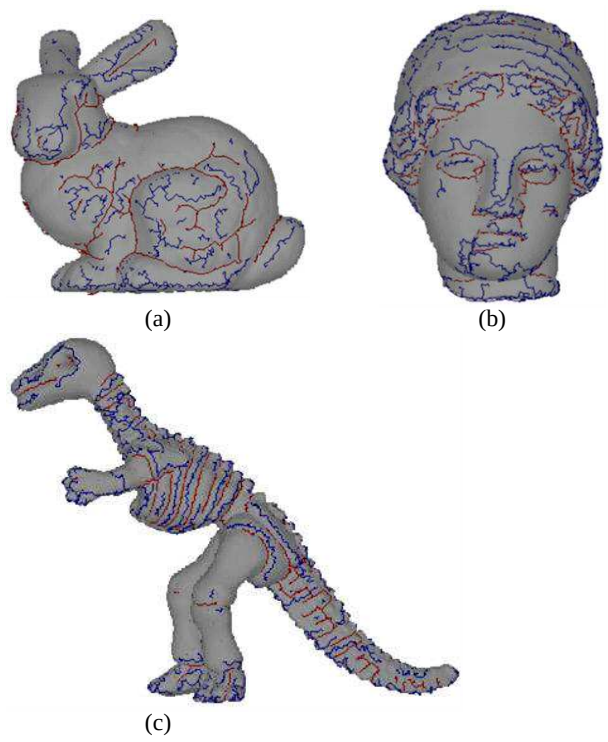


Figure 7: Results on Crest lines. Ridges are blue and valleys are red. (a) Stanford Bunny, (b) Igea and (c) Dinosaur.

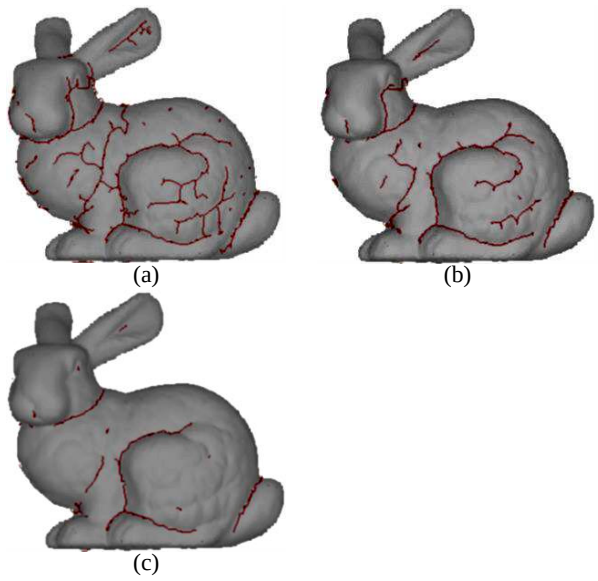


Figure 8: Sliding threshold M. (a) M=0.01, (b) M=3, (c) M=6.

6 CONCLUSIONS AND FUTURE WORK

We have presented a complete, simple and automatic method for extracting crest lines from surfaces represented by point clouds. Initially, we convert the point cloud to an undirectional graph using the k-closest neighbor operation. This method approximates normal curvature on every point, identifies every crest point and then joins them using region growing. Minimum spanning tree is used for thinning the crest graphs. It is quite general and can be used on any type of point cloud surfaces.

We plan to utilize crest lines over point clouds to achieve feature preserving point cloud simplification and to assist in the development of new adaptive triangulation techniques, among others.

References

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. *IEEE Visualization 2001*, 21–28.

ANDRESEN, P., NIELSEN, M., AND KREIBORG, S. 1998. 4D shape-preserving modelling of bone growth. MICCAI.

ARYA, S., MOUNT, D. M., NETANYAHU, N. S., SILVERMAN, R., AND WU, A. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *JACM: Journal of the ACM 45(6)*, 891–923.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1993. *Introduction to Algorithms*. MIT Press.

DECLERCK, J., SUBSOL, G., THIRION, J. P., AND AYACHE, N. 1995. Automatic retrieval of anatomical structures in 3d medical images. In *CVRMed'95 Lecture Notes in Computer Science*, N. Ayache, Ed., vol. 905. Springer-Verlag, 153–162.

FARIN, G. 2001. *Curves and Surfaces for Computer Aided Geometric Design, 5th.* Morgan-Kaufmann.

FLEISHMAN, S., COHEN-OR, D., ALEXA, M., AND SILVA, C. T. 2003. Progressive point set surfaces. *ACM Trans. Graph. 22*, 4, 997–1011.

GUÉZIEC, A., AND AYACHE, N. 1992. Smoothing and matching of 3d space curves. European Conference on Computer Vision, 620–629.

GUÉZIEC, A. 1993. Large deformable splines, crest lines and matching. IEEE 4th International Conference on Computer Vision, 650–657.

GUMHOLD, S., WANG, X., AND MACLEOD, R. 2001. Feature extraction from point clouds. 10th International Meshing Roundtable, 293–305.

HAMANN, B. 1993. Curvature approximation for triangulated surfaces. In *Geometric Modelling*, F. *et al*, Ed. Springer Berlin, 139–153.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. ACM Siggraph, 71–78.

HUBELI, A., AND GROSS, M. 2001. Multiresolution feature extraction from unstructured meshes. IEEE Visualization.

KHANEJA, N., MILLER, M., AND GRENANDER, U. 1998. Dynamic programming generation of curves on brain surfaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence 20(11)*, 1260–1265.

LENGAGNE, R., PASCAL, F., AND MONGA, O. 1996. Using crest lines to guide surface reconstruction from stereo. 13th International Conference on Pattern Recognition.

MONGA, O., ARMANDE, N., AND MONTESINOS, P. 1997. Thin nets and crest lines: Application to satellite data and medical images. *Computer Vision and Image Understanding 67(3)*, 285–295.

OHTAKE, Y., AND BELYAEV, A. 2001. Automatic detection of geodesic ridges and ravines on polygonal surfaces. *The Journal of Three Dimensional Images 15(1)*, 127–132.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. *ACM Siggraph, to appear*.

PAULY, M., GROSS, M., AND KOBBELT, L. 2002. Efficient simplification of point-sampled geometry. IEEE Visualization.

PAULY, M., KOBBELT, L., AND GROSS, M. 2002. Multiresolution modeling of point-sampled geometry. Tech. rep., ETH Zurich, September.

PAULY, M., KEISER, R., AND GROSS, M. 2003. Multi-scale feature extraction on point-sampled surfaces. vol. 22(3) of *Eurographics*.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels:surface elements as rendering primitives. ACM Siggraph Conf. Proc.

RUSINKIEWITZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. ACM Siggraph Conf. Proc.

STYLIANOU, G. 2003. *Crest Lines and Their Applications with Emphasis on Anatomical Surfaces*. PhD thesis, Arizona State University, AZ, USA.

STYLIANOU, G. 2004. A feature based method for rigid registration of anatomical surfaces. In *Geometric Modelling for Scientific Visualization*, G. Brunett, B. Hamann, and H. Mueller, Eds. Springer-Verlag.

THIRION, J., AND GOURDON, A. 1996. The 3d marching lines algorithm. *Graphical Models and Image Processing 58(6)*, 503–509.

ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2001. Surface splatting. ACM Siggraph Conf. Proc.

ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2002. Photoshop 3d: An interactive system for point-based surface editing. ACM Siggraph Conf. Proc.

About the author

Georgios Stylianou is a lecturer in the Department of Computer Science at Cyprus College, Cyprus. He received his PhD in Computer Science from Arizona State University, USA, in 2003 and his BSc in Computer Science from the University of Cyprus in 1998. His research interests lie in the area of geometric modelling and include parametrization and multi-resolution methods, feature extraction, deformation and registration.

Yiorgos Chrysanthou is an Assistant Professor at the Computer Science Department, University of Cyprus since 2001. Before that he was at the University College London, as a Research Fellow (1996-1998) and a Lecturer (1998-2001), part of the Computer Graphics and Virtual Environments group. He studied for his BSc in Computer Science and Statistics and for his PhD in Computer Graphics, at Queen Mary and Westfield College, University of London. His current research interests are in the broader area of Computer Graphics and include real-time rendering, visibility, crowd rendering and simulation, virtual and augmented reality and applications to cultural heritage.