Extracting CAD features from point cloud cross-sections

Ioannis Kyriazis
University of Ioannina, Greece
kyriazis@cs.uoi.gr

Ioannis Fudos
University of Ioannina, Greece
fudos@cs.uoi.gr

Leonidas Palios
University of Ioannina, Greece
palios@cs.uoi.gr

ABSTRACT

We present a new method for extracting features of a 3D object targeted to CAD modeling directly from the point cloud of its surface scan. The objective is to obtain an editable CAD model that is manufacturable and describes accurately the structure and topology of the point cloud. The entire process is carried out with the least human intervention possible. First, the point cloud is sliced interactively in cross sections. Each cross section consists of a 2D point cloud. Then, a collection of segments represented by a set of feature points is derived for each slice, describing the cross section accurately, and providing the basis for an editable feature-based CAD model. For the extraction of the feature points, we exploit properties of the convex hull and the Voronoi diagram of the point cloud.

Keywords: Reverse Engineering, processing point clouds, feature points, Voronoi diagram

1 INTRODUCTION

2005January 31 – February 4

Many applications in manufacturing, medicine, geography, design, and entertainment require scanning of rather complex three-dimensional objects for the purposes of incorporating them into a computer-based or computer-aided processing system, a technique commonly known as *Reverse Engineering* or *Digital Reconstruction* [Vara97, Thom99, Benk01, Thom96]. 3D scanning of a solid object yields a representation that consists of the coordinates of several points of the surface of the object. Advanced measuring techniques have been developed that produce a large amount of points lying on the object surface. Such a point set representing the boundary of a three-dimensional object is often called a *point cloud*. Point clouds do not include structural or connectivity information, and therefore they do not provide editable models.

We present a method for dividing the point cloud into several slices (cross-sections), which we process separately to extract local structural information. This information is used to detect local features of the object. Such features describe holes, extrusions or protrusions on the object, symmetrical or similar parts, or flipped and arbitrarily transformed versions of already known features. More specifically, our method uses a subset of the point cloud each time, which is subsequently used to extract a feature locally [Gumh01]. This is accomplished by dividing the point cloud into several slices. The points of each slice can be considered to be coplanar, and so we process each slice as a 2D set of points allowing us to develop very efficient and accurate local feature extraction techniques. The set of points of each slice are processed with efficient algorithms and are represented with a sequence of feature points derived with advanced computational geometry based techniques. We call this sequence of feature points *feature polyline*. A closed cubic B-Spline curve is then computed, which interpolates the feature points. The local per slice feature representation is then combined with information provided from several adjacent slices, to reconstruct the global structure and morphology of the object.

In a nutshell, our paper makes the following technical contributions:

- Presents an interactive technique for segmenting the point cloud into slices. This process is supported by optimization algorithms for correct placement and user-intuitive visualization techniques.
- Introduces a fast technique for detecting characteristic points that describe the shape of a slice based on iterative application of the convex hull and Voronoi diagram algorithms.
- Establishes a novel theoretical formulation for characterizing structural properties of 2D point sets.
- Provides an editable representation of the 2D point sets with the use of closed cubic B-Spline curves.

The rest of this paper is structured as follows. Section 2 presents related work. Section 3 presents the interactive slicing technique. Section 4 introduces the theoretical formulation for morphology analysis and presents a

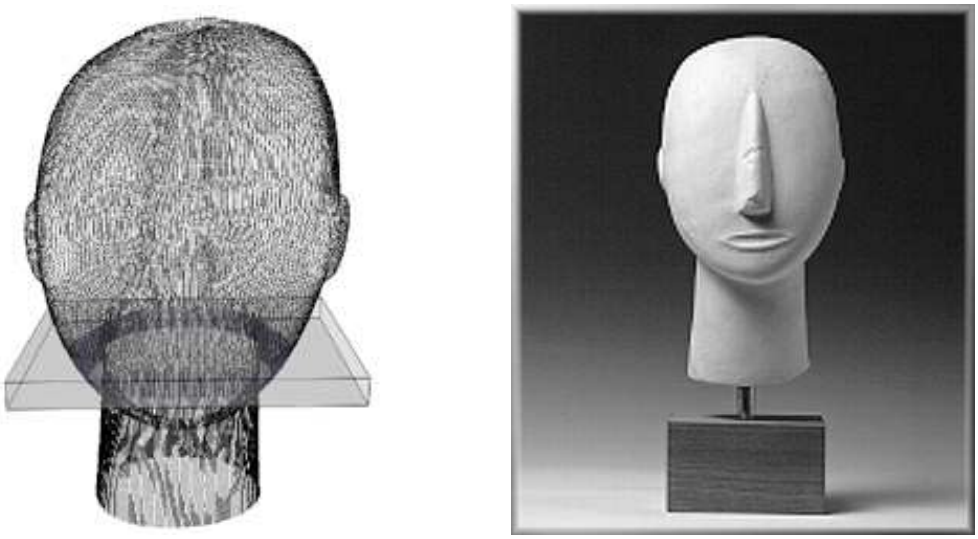

Figure 1: A point cloud of a cycladean idol, and the original model. The point cloud was acquired using a base Scanny 3d color laser scanner [Scan08].

convex hull and Voronoi-diagram based method to detecting feature point sets. Section 5 describes the B-Spline curve interpolation. Section 6 presents a performance analysis of the methods and Section 7 offers a summary of our results.

2 RELATED WORK

Several methods have been developed that extract features from a point cloud. Some of these methods are applied on mechanical objects, and others on freeform objects. While mechanical objects are easy to describe with a standard a set of features, representing freeform objects necessitates the expansion of the usual repertoire of features and operation with innovative constraint-based features. Jeong et al. [Jeon02] use an automated procedure to fit a hand-designed generic control mesh to a point cloud of a human head scan. A hierarchical structure of displaced subdivision surfaces is constructed, which approximates the input geometry with increasing precision, up to the sampling resolution of the input data. Sithole and Vosselman [Sith03] have developed a method for detecting urban structures in an irregularly spaced point-cloud of an urban landscape. Their method is designed for detecting structures that are extensions to the bare-earth (e.g., bridges, ramps), and it involves a segmentation of a point-cloud followed by a classification. Au and Yuenb [Au99] use a method that fits a generic feature model of a human torso to a point cloud of a human torso scan. The features are recognized within the point cloud by comparison with the generic feature model. This is achieved by optimizing the distance between the point cloud and the feature surface, subject to continuity requirements. This is a powerful approach when we have a priori knowledge of the set of features. Amenta et al. [Amen98] proposed the crust algorithm, which combines the point cloud with the vertices of the Voronoi diagram, and computes the Delaunay tetrahedralization of the combined point set. The triangles where all vertices are sample points (not Voronoi vertices) are considered to form the object surface. Attene and Spagnuolo [Atte00] use properties of geometric graphs. The Euclidean minimum spanning tree is used as a constraint during the sculpturing of the Delaunay tetrahedralization of the data set, and in addition another constraint is used, the so-called Extended Gabriel Hypergraph (EGH). These approaches are very interesting and with many application in computer graphics.

These methods however provide a triangular representation with the capability of only local editing by altering interactively the positioning of triangle vertices. Such models are not appropriate for editing and remanufacturing in the context of computer aided design.

Fayolle et. al. [Shap04] propose a method which helps to fit existing parameterized function representation (FRep) models to a given dataset of 3D surface points. Best fitted parameters of the model are obtained by using a hybrid algorithm combining simulated annealing and Levenberg-Marquardt methods. Ohtake et. al. [Ohta03] construct surface models using piecewise quadratic functions that capture the local shape of the surface, and weighting functions that blend together the local shape functions. These works process the entire 3D cloud to detect the object's constructive logic.

Related work on deformable models is described in [Terz87, Meta93, Mill91, Mall95, McIn96]. These methods are quite general and have been applied in determining contours. To our knowledge there is no efficient method developed for extracting features based on deformable models in this context. However this is a promising direction that requires further research.

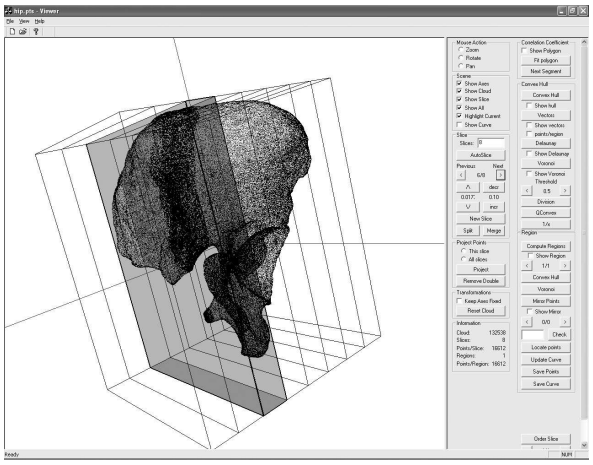

Figure 2: A point cloud can be sliced in any desired direction. Slice thickness is determined adaptively (point cloud of a hip bone [Cybe99]).

3 SLICING THE POINT CLOUD

Processing the point cloud is performed in two important steps:

- choosing an appropriate slicing direction and
- determining the proper thickness of a slice.

The direction along which we choose to divide the object in slices may influence the process of feature extraction and the resulting model. Thus, a major issue in point cloud segmentation is determining the slicing direction. To illustrate this, consider an object that contains a cylindrical hole. If the slicing direction matches the cylinder axis, the points of the slice located near the hole would form a circular region. If the slicing direction is different than the cylinder axis, the points of the slice would form an elliptical region or even a single line segment.

The effectiveness of the editability of the resulting CAD model also depends on the selection of the slicing direction. To this end we can either align interactively the object to the desired direction, or seek automatically a transformation of the object that minimizes a target function, e.g. PCA [Joll02]. We use a combination of automated and interactive methods.

Another key issue is to determine the proper thickness for a slice. When a slice is too thick, the points that belong to it will not provide useful information, as we might get many features tangled together. To avoid this we can split it into two new slices with reduced thickness. On the other hand, if the points of two adjacent slices carry almost the same information, or if the points of a slice are not enough to describe this part of the object resulting in discontinuities, we can merge the two slices into one slice with increased thickness. The thickness of a slice may also differ from slice to slice, if we require more detailed processing in some parts of the point cloud than the rest.

To avoid performing a morphological analysis of the 3D point cloud in the preprocessing phase, we have chosen to initially slice the point cloud in slices of equal thickness, by setting the slice thickness manually. We may also determine the ideal slice thickness adaptively, as described in [Wu04, Ma04, GHLi02, Star05]. The distance between two slices can also be a parameter, but as we do not want to omit any information from points located between two slices, we set each slice to start exactly where the previous slice ends (and thus there is no empty space between adjacent slices).

Once we have divided the point cloud into slices, the points that belong to each slice are projected on a plane that is perpendicular to the slicing axis. After this preprocessing we may use 2D techniques for processing the points of the slice. Figure 2 illustrates an example of a sliced point cloud of a hip bone from Cyberware [Cybe99], while Figure 3 illustrates the points of a slice, which are projected to the highlighted plane. Figure 2 also depicts a snapshot of the user interface of the prototype that we have developed to evaluate the effectiveness of method.

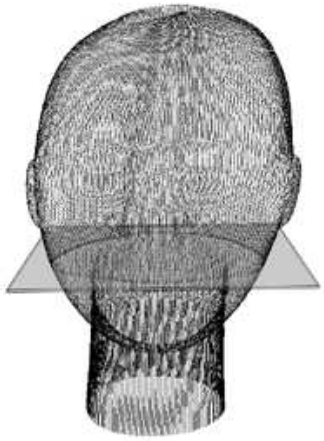

Figure 3: A slice of the cloud is highlighted, and the points of this slice are projected on a plane that is perpendicular to the slicing direction.

4 IDENTIFYING FEATURE POINTS

In reverse engineering, a point cloud usually consists of a very large amount of points, depending on the size and shape of the prototype object, and also on the accuracy that was used to scan the object. The large amount of points makes it difficult to process this raw information. Thus, we need to reduce the number of points in the cloud while retaining most of the topology implied provided by these points.

If the points of a slice form a 2D shape that is convex, then we simply compute the convex hull of the points. This will accurately describe the shape with a polyline consisting usually of much fewer vertices than the points of the slice. For example, if the points of a

slice lie on a rectangle, the convex hull consists of only four vertices and four edges, while the slice may consist of thousands of points.

However, if the shape implied by the points in a slice is not convex then fitting a polyline to these points needs additional information beyond the convex hull. We start by computing the convex hull [Barb96] of the points as shown in Figure 4, to identify some initial feature points. We partition the points in regions, one for each line segment of the convex hull.

Some of these regions may consist exclusively of points very close to the convex hull (convex regions), while other regions contain of points located far from it (concave regions). Before treating these concave regions any further, we need to check wether each point is assigned to the proper region, since there may be cases in which a point is closest to one region, but belongs to another region, for example the one that is located on the opposite side of the closed feature polyline(e.g. see Figure 5). To avoid having such erroneous assignments we define:

Definition 1 *A point p_N is a* neighbor point *of a point p if it lies within distance smaller or equal to some characteristic constant ε.*

ε is called the *neighborhood radius* which is a characteristic of the point cloud and is derived from statistically processing the topology of the slice.

Definition 2 *We call* separability tolerance *$d > \varepsilon$ of the point cloud, the minimum number with the property that for any pair of points p_1, p_2 that belong to different non-adjacent regions it holds $||p_1 - p_2|| \leq d$.*

d is a characteristic of the point cloud derived from statistically processing the topology of the cross-section.

Then we have,

Definition 3 *The* shortest neighbor path *between two points s_1 and s_n of the point cloud slice is a sequence of points $[s_1, s_2, \ldots, s_n]$ such that each point is neighbor to the next and*

$$\sum_{i=1}^{n-1} ||s_i - s_{i+1}||$$

is minimized.

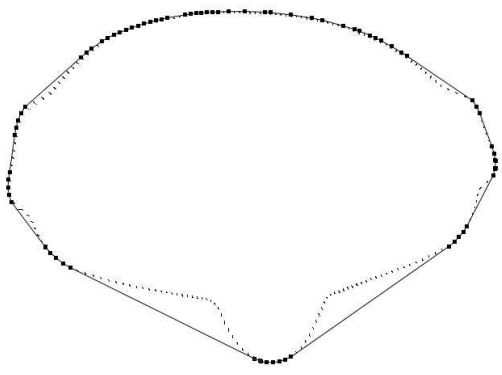

Figure 4: The convex hull of the slice points in 2D. The vertices of the convex hull are identified as feature points for this slice (cycladean idol cloud [Scan08]).

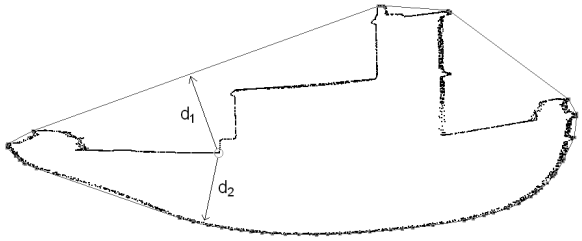

Figure 5: The points in the highlighted area (small circle) are located closer to a region other than they should be assigned to $(d_2 < d_1)$. We use the information of their neighboring points to assign them to the correct region (point cloud of a boat [Cybe99]).

Then given a point sequence $P = [p_1, p_2, \ldots p_k]$ that segments the slice into regions $[r_1, r_2, \ldots r_k]$ we calculate the shortest neighbor paths between each adjacent pair of points $(p_i, p_{(i+1) \bmod n})$.

Definition 4 *The* initial seed *of a region r_i is the shortest path between p_i and $p_{(i+1) \bmod n}$.*

Finally, we assign each point q of the point cloud to a region r_i such that the initial seed of r_i contains a point s_j that minimizes the shortest neighbor path length from q.

Definition 5 *For a point cloud point q the* region of q *is defined as the region r_i such that the initial seed of r_i contains a point s_j that minimizes the shortest neighbor path length from q.*

Now that we have ensured that each point is associated to the correct region, we can treat the concave regions further by computing the Voronoi diagram of their points, and using a property called the *largest empty circle* [De B97, ORou98].

The idea is to have a circle of variable radius and throw it towards the point cloud from a given direction (the general idea includes a sphere on the three-dimensional space). When it touches a point of the region, this point is fixed and the circle continues to move around it, until a second point is touched. When the circle touches the second point, it is also fixed, and the only free variable is the radius, which now starts decreasing, until the circle touches a third point. The third point is also fixed, and the circle cannot move any more. The three points of contact are added to the feature point set. The center of the circle is one of the Voronoi vertices of the region.

Thus, if we compute the Voronoi diagram for the point cloud region each Voronoi vertex located outside the point cloud area is a candidate center for a largest empty circle that is touching three or more region points. We can use this property for detecting empty largest circles that have their center on a Voronoi region that is far from the point cloud region and towards the outside. Then the feature points are the points of contact of such largest empty circles.

The number of Voronoi vertices that can be used as centers for largest empty circles is still large, and we have to choose those vertices that will provide suitable feature points. We do not look for Voronoi vertices located very close to the point cloud, because they would provide feature points located very close to each other, which can be useful only in cases where we need increased detail. We do not look for Voronoi vertices located very far from the points of the cloud either, because they would provide feature points located far from each other, leading to lower detail results. Also, we do not look for Voronoi vertices located close to each other, because they would provide the same feature points, or feature points very close to each other as in the first case. Furthermore, Voronoi vertices on the wrong side of the point cloud are also being excluded, because we are interested in feature points on the boundary of the point cloud. What we need is to choose several Voronoi vertices evenly distributed along the region points, at an intermediate distance from the region points. Figure 6 (right) illustrates the candidate Voronoi vertices for two concave regions.

By combining the initial convex hull with the feature points provided by the selected Voronoi vertices of each region, we get a feature polyline that interpolates the points of the slice adequately in most regions. We can perform this operation repeatedly for the rest of the regions until all regions consist of points that are located near the fitting polyline (for a termination criterion see for example [Said02]). The final result of the fitting feature polyline is illustrated in Figure 7 (right).

The method is summarized in Algorithm 1. Note that L is a plane perpendicular to the slicing axis.

The use of the Voronoi diagram and the largest empty circle for identifying feature points is also known as the rotating ball technique [Bern99], and the feature points form the so called alpha shape of the point set [Edel94].

Input: a set P of points, Slice i
Output:an ordered set F_i of feature points
$(P_i^{(3D)}, L) = slice(i, P)$
$P_i = project(P_i^{(3D)}, S)$
$F_i = qconvex(P_i)$
$F_{ij} = \emptyset$
repeat
  for each region P_{ij} of F_i **do**
    if $avg\_dist(P_{ij}, F_{ij}) > e$ **then**
      $V_i = qvoronoi(P_{ij})$
      $V_{candidate} = V_i -$ excluded Voronoi vertices
      $F_{ij} = largest\_empty\_circle(V_{candidate}, P_{ij})$
  $F_i = F_i \cup F_{ij}$
until $F_{ij} \neq \emptyset$
return F_i

Algorithm 1: The algorithm for the feature point extraction.

5 INTERPOLATING FEATURE POINTS WITH CUBIC B-SPLNE CURVES

The feature polyline we have extracted so far may provide useful information concerning the topology of the 2D point set, but it does not include smoothness requirements. As discussed in [Lee99], it is a common practice to use B-Spline curves, because they are easy to compute and capable of representing adequately most 3D objects. Thus we choose to interpolate a closed cubic B-Spline curve through the feature points of the cross-section.

We employ curves of degree 3, because it is the lowest degree satisfying second-order continuity, and at the same time ensuring that we have G^1 continuity. The knot vector, the parameter values and the control points of the interpolating curve are determined according to the method described in [Lee99].

Once we have acquired the curve that describes the cross-section, we proceed to the next cross-section and repeat the same process, until all cross-sections have been processed, and the point cloud is described by a sequence of B-Spline contours. Figure 8 illustrates an example of such a sequence.

So far, we have managed to represent the point cloud in an editable form, so that a designer may apply modifications to the 3d model.If the point cloud is a scan of a mechanical object, CAD features such as pockets, grooves, gears, holes could be extracted with further processing of the curve set. This would allow for higher level modifications to the model. But if the point cloud is a scan of a freeform object, such CAD features are not present, and the designer can only use the curve set for modifications. In this paper we focus on the use of cross sectional features and we may detect sub features such as pockets and holes by further post processing. Future work also includes using a skinning technique to reconstruct the surface between adjacent curves.

6 PERFORMANCE ANALYSIS

It is not possible to derive an exact evaluation of the complexity of our approach since this depends on the shape of the object. Thus in this section we provide an estimation of the expected complexity as a function of

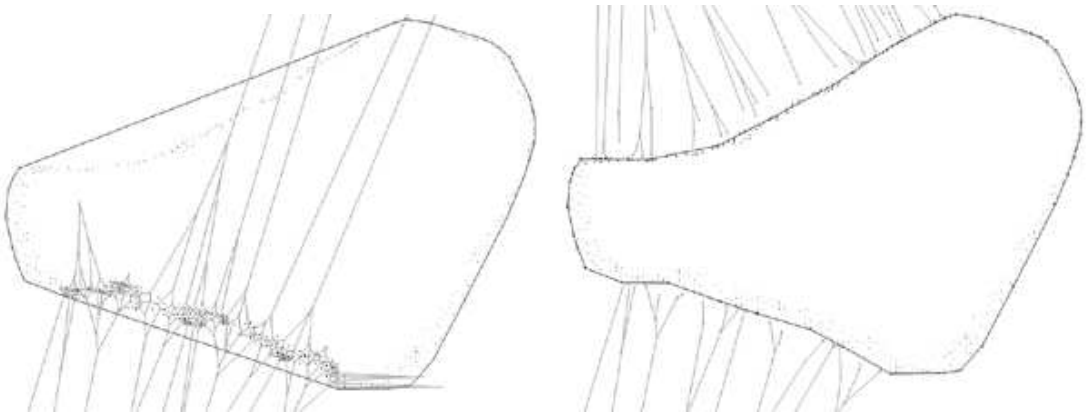

Figure 6: (Left)Voronoi vertices that lie reasonably far from the point cloud are used to identify the next set of feature points of this region. (Right) Only Voronoi vertices that are located outside the 2D area of interest are used for feature extraction. Here, this is illustrated for two regions (cross-section of the hip point cloud [Cybe99]).

Figure 7: For the feature polyline to fully describe a region, several iterations may be required. (Left) The first iteration of our method applied on a region. (Center) The second iteration for the same region. (Right) After the second iteration the region is described as expected.

the number of points in the point cloud. The complexity is measured in point operation as we have a large number of points in the point cloud and most of the processing is performed point-wise.

Before we apply our method to a point cloud, three standard steps are required, which require $O(n)$ point operations each (where n is the number of points in the cloud). These are (a) loading the cloud into memory, (b) slicing the cloud, and (c) projecting the slice points on the slice. Slicing the cloud requires $O(n)$ because each point has to be assigned to a slice, so the whole cloud has to be processed, regardless of the number of slices. The same applies for projecting the points to a slice.

The convex hull of each slice requires $O(n_i \log n_i)$ operations (n_i being the number of points in slice i). But since we compute the convex hull for all slices, it sums up to $O(\sum_{i=1}^{s} n_i \log n_i)$ where s is the number of slices, which is bounded by $O(n \log n)$, since $\sum_{i=1}^{s} n_i = n$.

At this point, we have to isolate the regions of the slice points according to the convex hull, and compute the Voronoi diagram only for those regions, which are not adequately described by the feature polyline. This step depends on the shape of the slice points, and may require computation for up to all regions (or for no region at all, if the points of the slice form a convex polygon).

Considering the case where we have to repeat the process for all regions, it would take $O(n_j \log n_j)$ point operations for the n_j points of region j. This means that we need $O(\sum_{j=1}^{r} n_j \log n_j)$ operations for slice i, where r is the number of regions, and $O(\sum_{i=1}^{s} \sum_{j=1}^{r} n_{ij} \log n_{ij})$ for all slices. This is also bounded by $O(n \log n)$, since

$$\sum_{i=1}^{s} \sum_{j=1}^{r} n_{ij} = n$$

One issue is the number of iterations required to fully fit the feature polyline to the slice points. It depends on the shape of the points, and in the worst case it may require up to $O(\log n_i)$ steps to process the n_i points of slice i, i.e. $O(\log n)$ for all slices. In practice, it usually takes only a constant number of steps.
To select the Voronoi vertices in all regions and all slices, it requires $O(n)$, and to identify the next feature points and update the feature polyline it requires a constant number of point operations.

To fully update the feature polyline in all slices and to identify all feature points we need $O(\log n)$ iterations of either $O(1)$, $O(n)$, or $O(n \log n)$ point operations. So, in

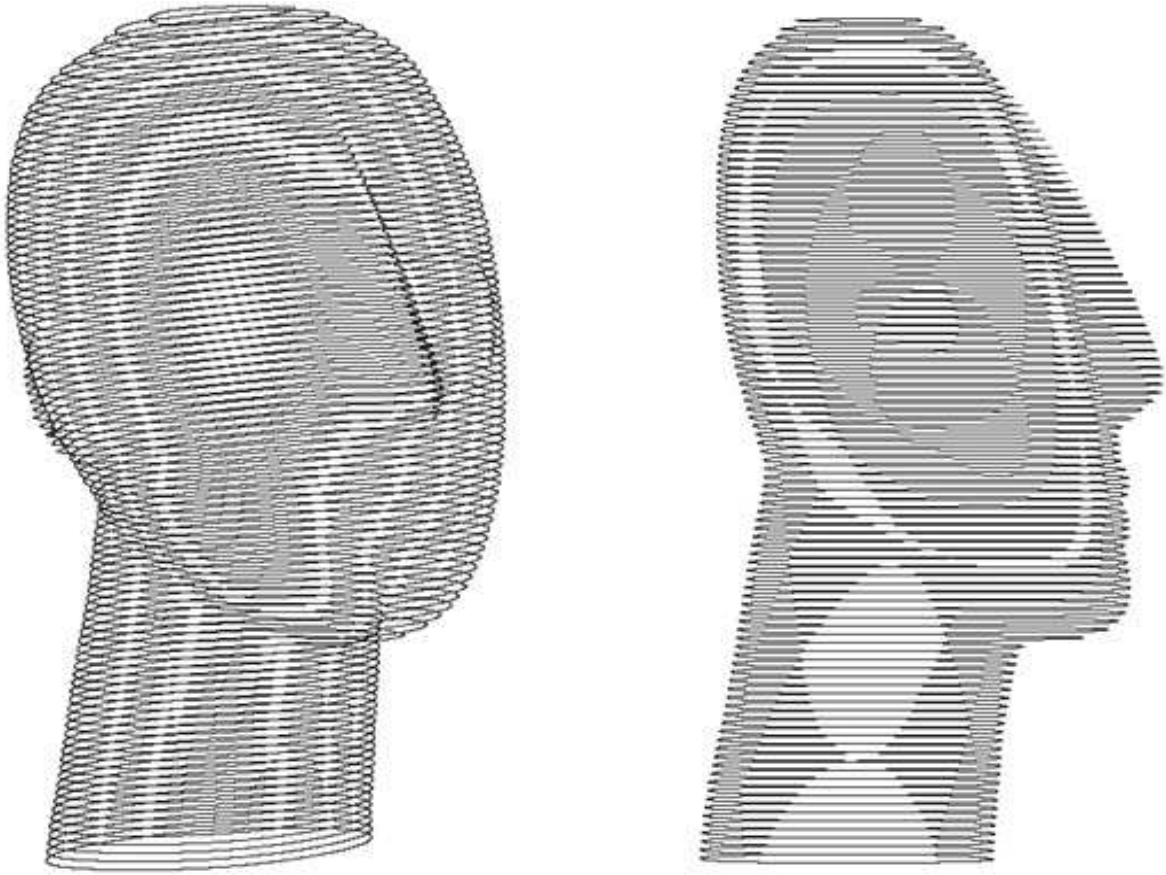

Figure 8: The curves of the entire cycladean idol point cloud (100 cross-sections).

conclusion, to derive descriptive feature point sets for all slices of the point cloud takes $O(n\log^2 n)$ time.

For the curve interpolation of a slice, the worst case includes all the points of the slice. Then we need $O(n_i)$ time to compute the knot vector, $O(n_i)$ for the parameter values, and $O(n)$ for the control points, since it requires solving a triagonal linear system of equations. In conclusion, for the whole point cloud to be processed, it requires $O(n\log^2 n)$ time.

7 SUMMARY

We have developed a method for representing cross sections (slices) of a point cloud, by identifying the a set of feature points for each cross-section. Future work includes building a solid model of the objects by using skinning techniques on the derived B-Spline cross-sections.

REFERENCES

[Amen98] N. Amenta, M. Bern, and M. Kamvysselis. "A New Voronoi-Based Surface Reconstruction Algorithm". *Computer Graphics*, Vol. 32, No. Annual Conference Series, pp. 415–421, 1998.

[Atte00] M. Attene and M. Spagnuolo. "Automatic surface reconstruction from point sets in space". *Computer Graphics Forum*, Vol. 19, No. 3, pp. 457–465, 2000.

[Au99] C. Au and M. Yuenb. "Feature-based reverse engineering of mannequin for garment design". *Computer-Aided Design*, Vol. 31, pp. 751–759, 1999.

[Barb96] C. Barber, D. Dobkin, and H. Huhdanpaa. "The Quickhull algorithm for convex hulls". *ACM Transactions on Mathematical Software*, Vol. 22, No. 4, pp. 469–483, Dec 1996. http://www.qhull.org.

[Benk01] P. Benko, R. Martin, and T. Varady. "Algorithms for Reverse Engineering Boundary Representation Models". *Computer-Aided Design*, Vol. 33, No. 11, pp. –851, 2001.

[Bern99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. "The ball-pivoting algorithm for surface reconstruction". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, pp. 349–359, 1999.

[Cybe99] Cyberware. "Cyberware Rapid 3D Scanners". 1999. http://www.cyberware.com.

[De B97] M. e. a. De Berg. *Computational Geometry, Algorithms and Applications*. Springer-Verlag, 1997.

[Edel94] H. Edelsbrunner and E. P. Mücke. "Three-dimensional alpha shapes". *ACM Trans. Graph.*, Vol. 13, No. 1, pp. 43–72, 1994.

[GHLi02] G.H.Liu, Y.S.Wong, Y.F.Zhang, and H.T.Loh. "Error-based segmentation of cloud data for direct rapid prototyping". *Computer-Aided Design*, Vol. 35, pp. 633–645, 2002.

[Gumh01] S. Gumhold, X. Wang, and R. MacLeod. "Feature Extraction from Point Clouds". In: *Proceedings of 10th International Meshing Roundtable, Newport Beach, CA*, pp. 293–305, Sandia National Laboratories, October 2001.

[Jeon02] W.-K. Jeong, K. Kahler, J. Haber, and H.-P. Seidel. "Automatic Generation of Subdivision Surface Head Models from Point Cloud Data". In: *In Proceedings Graphics Interface 2002*, pp. 181–188, 2002.

[Joll02] I. Jolliffe. *Principal Components Analysis*. Springer, 2nd edition Ed., 2002.

[Lee99] K. Lee. *Principles of CAD/CAM/CAE Systems*. Addison Wesley, pearson international edition Ed., 1999.

[Ma04] W. Ma, W.-C. But, and P. He. "NURBS-based adaptive slicing for efficient rapid prototyping". *Computer-Aided Design*, Vol. 36, pp. 1309–1325, 2004.

[Mall95] R. Malladi, J. Sethian, and B. Vemuri. "Shape modeling with front propagation: A level set approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 2, pp. 158–175, February 1995.

[McIn96] T. McInerney and D. Terzopoulos. "Deformable models in medical image analysis: a survey". *Medical Image Analysis*, Vol. 1, No. 2, pp. 91–108, 1996.

[Meta93] D. Metaxas and D. Terzopoulos. "Shape and nonrigid motion estimation through physics-based synthesis". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 6, pp. 580–591, June 1993.

[Mill91] J. Miller, D. Breen, W. Lorensen, R. O'Bara, and M. Wozny. "Geometric deformed models: a method for extracting closed geometric models from volume data". In: *In SIGGRAPH, Computer Graphics Proceedings*, pp. 217–226, ACM SIGGRAPH, July 1991.

[Ohta03] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. "Multi-level partition of unity implicits". *ACM Trans. Graph.*, Vol. 22, No. 3, pp. 463–470, 2003.

[ORou98] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[Said02] M. A. Said. "Polyline Approximation of Single-Valued Digital Curves Using Alternating Convex Hulls". *Computer Graphics and Geometry*, Vol. 4, pp. 75–99, 2002.

[Scan08] Scanny3d. "Scanny 3d laser scanners". 2008. http://www.scanny3d.com.

[Shap04] "Shape Recovery Using Functionally Represented Constructive Models". In: *SMI '04: Proceedings of the Shape Modeling International 2004*, pp. 375–378, IEEE Computer Society, Washington, DC, USA, 2004.

[Sith03] G. Sithole and G. Vosselman. "Automatic Structure Detection in a Point-Cloud of an Urban Landscape". In: *Remote Sensing and Data Fusion over Urban Areas, 2nd GRSS/ISPRS Joint Workshop*, pp. 67–71, 2003.

[Star05] B. Starly, A. Lau, W. Sun, W. Lau, and T. Bradbury. "Direct slicing of STEP based NURBS models for layered manufacturing". *Computer-Aided Design*, Vol. 37, pp. 387–397, 2005.

[Terz87] D. Terzopoulos, A. Witkin, and M. Kass. "Symmetry-seeking models and 3d object reconstruction". *International Journal of Computer Vision*, Vol. 1, No. 3, pp. 211–221, October 1987.

[Thom96] W. Thompson, H. de St. Germain, T. Henderson, and J. Owen. "Constructing high-precision geometric models from sensed position data". In: *Proc. ARPA Image Understanding Workshop*, February 1996.

[Thom99] W. Thompson, J. Owen, H. de St. Germain, S. Stark, and T. Henderson. "Feature-Based Reverse Engineering of Mechanical Parts". *Trans. Robotics and Automation*, Vol. 15, No. 1, pp. 57–66, 1999.

[Vara97] T. Varady, R. Martin, and J. Cox. "Reverse Engineering of Geometric models, An Introduction". *Computer-Aided Design*, Vol. 29, No. 4, pp. 255–269, 1997.

[Wu04] Y. Wu, Y. Wong, H. Loh, and Y.F.Zhang. "Modelling cloud data using an adaptive slicing approach". *Computer-Aided Design*, Vol. 36, pp. 231–240, 2004.