www.cadanda.com
Computer-Aided Design AND APPLICATIONS™

Building Editable Free-form Models from Unstructured Point Clouds

Ioannis Kyriazis[1] and Ioannis Fudos[2]

[1]University of Ioannina, kyriazis@cs.uoi.gr
[2]University of Ioannina, fudos@cs.uoi.gr

ABSTRACT

We present a novel approach to reconstructing the surface of a 3D object using a 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 point cloud is sliced into cross sections. Each cross section is represented as a 2D point set. Then, a collection of interpolating curves is computed that describes each cross section accurately, using computational geometry methods on the corresponding point set. The organized point set for the surface reconstruction is obtained from these curves. Freeform transformations may also be applied, allowing high level global editing of the final object.

Keywords: surface reconstruction, cross sections, transformations.
DOI: 10.3722/cadaps.2013.877-888

1 INTRODUCTION

Surface reconstruction is widely used in a broad spectrum of domains such as computer graphics, CAD, reverse engineering and medical imaging. The process is brought down to obtaining a point cloud, i.e. a set of points that usually lie on the boundary of a 3D object, using one of several 3D point acquisition methods available (laser scanning, photogrammetry, CT scan) [3], and then process this point cloud to extract a CAD model of the object surface. The point cloud may be acquired for example with the use of a 3D laser scanner [6], or by identifying feature points on multi-camera images [8],[16], or even computerized tomography when it comes to medical applications [15],[20]. For processing the point cloud, various methods have been proposed, which include slicing the point cloud into cross-sections [15], or patches [19], or treating the point cloud as a whole [1].

Our method uses the approach of slicing the point cloud into cross sections, and treating each cross section as an individual point set in 2D. A closed curve of continuity G^0 (i.e. a polyline) is initially computed, and subsequently a G^1 B-Spline curve is constructed, which interpolates the polyline and provides a smooth boundary representation. The contours of all cross sections are combined and a number of points are selected (depending on the level of detail that is requested) on the B-Splines which are subsequently used for creating a mesh that represents the reconstructed surface with an editable model. A series of editing operations may be performed on the model, to produce several variations of the original model, while maintaining certain attributes that capture user intent. After a set of editing operation is applied, a re-computation of the B-splines and the mesh is carried out.

The objective of this work is to provide a novel paradigm for creating an editable model based on B-spline contours without the use of parametric surface patches that is capable of supporting arbitrary tessellations.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 presents a technique for slicing the point cloud into cross sections. Section 4 provides an overview of how to compute a feature polyline for each cross section. In Section 5 we introduce a technique for deriving a new interpolating cross section using the information of existing cross sections. In Section 6 we offer details on computing planar B-spline curves fitting the polyline data with G^1 continuity. Section 7 introduces a method for reconstructing the surface of the object based on the cross section B-splines. Section 8, presents a novel paradigm for applying global modification operations to the derived model. Section 9 provides implementation considerations and some examples of applying our editing paradigm. Finally, Section 10 offers conclusions and future work directions.

Part of this work has also been discussed in [11], which describes an earlier stage of the development of our method. For sections 3, 4 and 6, in depth technical details have been provided in [11], so a brief description is given here. Section 5 improves the quality of the results of section 4, so that the output curves of section 6 produce better results in the reconstruction of the surface in section 7. Sections 7 and 8 were discussed as future work in [11], and are presented in this paper.

2 RELATED WORK

Several methods have been developed that extract features from a point cloud or a set of images. Some of these methods are appropriate for mechanical objects, and others for free-form objects. While mechanical objects can be represented via a standard set of features, representing free-form objects necessitates the expansion of the usual repertoire of features and operations with constraint-based features that are computationally expensive to solve.

Jeong et al. [9] use an automated process 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.

Au and Yuen [2] use a method to fit 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 minimizing 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. [1] 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. These approaches are very interesting and have found several applications in computer graphics.

Ohtake et. al. [14] 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.

Geng et al [8] have developed a method for rapid and accurate face recognition purposes, which uses a unique 3D camera (the 3D FaceCam) that combines multiple imaging sensors within a single compact device to provide instantaneous, ear-to-ear coverage of a human face. Thus, multiple 3D views are used to provide detailed and complete 3D coverage of the entire face.

Weng et al [20] propose a surface rendering method using optical flow, an apparent motion in the image plane produced by the projection of real 3D motion on a 2D image. They obtain an accurate 3D model of the object, by extracting the surface information from 3D motion. Their method is suitable for the reconstruction of 3D models from ultrasound medical images as well as other computed tomograms.

Peir et al [15] reconstruct the shape of geometries derived from a set of medical images representing planar cross sections of the object. The reconstruction is based on the interpolation of an

implicit function through a set of points obtained from the segmentation of the images. This approach allows for smooth interpolation between sections of different topology. The boundary of the object is an iso-surface of the implicit function that is approximated by a triangulation extracted by the method of marching cubes.

A survey on methods for reconstructing surfaces from unorganized point sets is also available in [13] where several known methods are evaluated.

These works process the entire 3D cloud as a whole or in parts, to detect the object's constructive logic. They provide a triangular representation with the capability of only local editing by altering interactively the positioning of triangle vertices. The main goal of these works is to reconstruct the surface for visualization purposes only. But there are cases where a designer wishes to manipulate their model using high level editing methods. The main contribution of our method is that it allows editing and remanufacturing of the model in the context of computer aided design.

3 COMPUTING THE CROSS-SECTIONS

Initially, the point cloud carries no information concerning the topology or the geometry of the object. It consists only of the 3D coordinates of the vertices. Little can be done for editing the object in this form, as the points are unorganized. At first, to obtain an organized point set for the object, we divide the point cloud into a number of cross sections. Each of these cross sections is defined as a thin slice of the point cloud, and the points that belong to this slice are treated later on as an individual 2D point set. The points are actually projected on the 2D plane of the slice.

To slice the point cloud into cross sections, we need to consider a number of user-specified parameters, i.e. the direction of the cross sections, the thickness of each cross section, and the distance between cross sections.

The proper slicing direction is related to the medial axis of the object. For simple objects with no protrusions, the medial axis consists of a simple line or curve. A direction following the average path of this curve usually satisfies the user requirements. For complex objects that have protrusions, the medial axis consists of a set of lines or curve that form the skeleton of the object. Such point clouds are broken into simpler parts, which are treated as independent point clouds, and each part is sliced according to its proper direction. For our test cases, to align the point cloud to the proper direction, we perform principal component analysis using the ALGLIB numerical analysis library [5]. The entire point cloud is then translated so that the principal axis matches the z-axis of the environment.

The thickness of each cross section also affects the quality of the resulting model. When we have thick cross sections, the projected 2D point set differ from the initial 3D point set significantly. When we have thin cross sections, the may contain too few points, which do not describe the slices properly. Objects with a rough surface are sliced into thin cross sections, so that each cross section describes the part of the object accurately. For objects with a smooth surface the cross sections are thicker, because if we take thin cross sections, we may end up with several adjacent cross sections that provide the same information. The thickness of the cross sections may vary from slice to slice, as some parts of the point cloud may require thinner cross sections than other parts.

Another property of the cross sections is the distance between two adjacent cross sections. There may be cases where the user requires some empty space between some cross sections. For example if the user intentionally omits parts of the point cloud from being processed, or if there are several cross sections that provide no useful information and may be merged into one cross section. The thickness of a cross section does not count as distance from a cross section to the next, although when projected to its corresponding plane, the points of the cross section are positioned at a certain distance from the points of the next cross section. The term 'distance' here refers only to empty space that is not included in the computations. However, if we choose to have non-zero distance between two cross sections, we should have in mind that any points located within the empty space will also be omitted from computations, resulting in the loss of potentially significant information.

In our implementation, the user may specify several options, such as the number of cross sections, the ability to merge cross sections or split a cross section in half. One or more cross sections may also be selected, even if they are not adjacent. This will be useful later on, when we perform modifications

on our model. The ability to define a new cross section between two existing cross sections is also discussed in detail in section 5. Fig. 1 shows an example of a point cloud that represents a Cycladic idol, and the acquired point cloud, which consists of 131517 points. For visualization purposes, the point cloud in Fig. 1(d) has been sliced in 25 cross sections only. For the actual computations the point cloud was sliced into 100 cross sections.

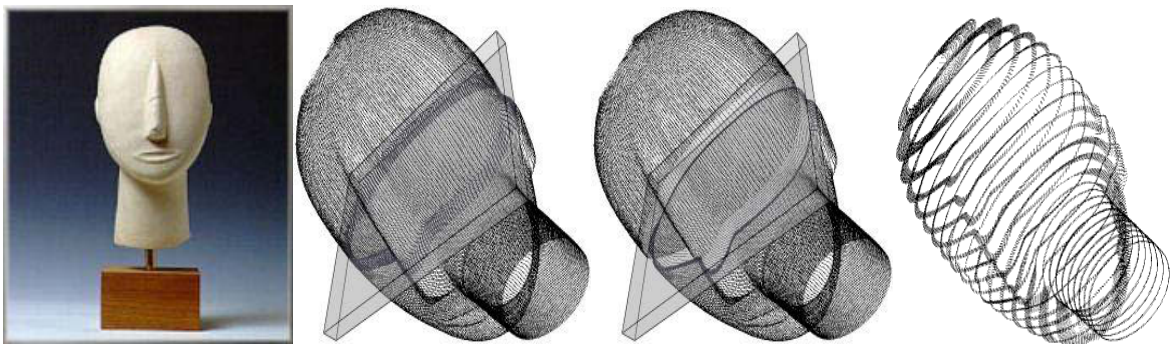

Fig. 1: The Cycladic idol is one of the most recognizable specimens of ancient Greek sculpture. (a) The actual object, (b) the point cloud of its surface scan, with one cross section highlighted, (c) the cross section is projected to the 2D plane, (d) the entire point cloud is sliced into cross sections, and the points of each cross section are projected to its corresponding 2D plane.

Another example is shown in Fig. 2 which is scan of a twisted drill bit. This point cloud consists of 1436231 points. Again for visualization purposes the point cloud in Fig. 2 (right) was sliced into 40 cross sections, while in the actual computations it was sliced into 200 cross sections.

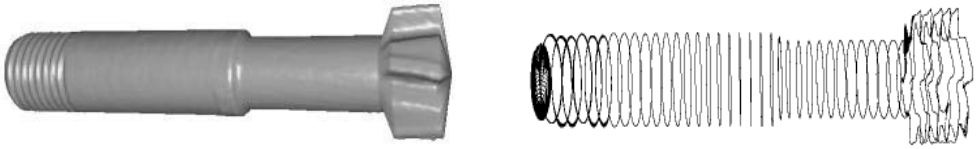

Fig. 2: A twisted drill bit: (Left) the actual object, (Right) the point cloud sliced into cross sections.

4 COMPUTING THE FEATURE POLY-LINE

Having computed the cross sections of the point cloud, we managed to provide some kind of organization to the point set, but we cannot perform high level modifications as there is no information available concerning the connectivity between the points. We need to extract information for the boundary of the point set in each cross section. The boundary will have the form of an ordered list that will be used later on to compute a smooth curve. The ordered list is also referred to as feature polyline, as the feature points form a polygon that is regarded as a sequence of parametric curves of degree one. When a cross section is represented with a parametric curve of higher degree, it provides the user with a smooth and flexible editable model.

To find the points that form the ordered list, which represents the boundary of the cross section, we use computational geometry methods and identify feature points that hold certain properties. We begin with the convex hull of the cross section. The points of the convex hull are boundary points and are immediately identified as feature points. If the cross section is convex the feature points from the convex hull will form the ordered list and this feature polyline will describe the cross section accurately. This means that no more computations are required for this cross section. But in most cases the cross section will not be convex, and the feature polyline will not describe the cross section accurately in all regions. For those regions, additional computations will be required, and the feature polyline will be updated with new feature points. The new feature points are identified using a Voronoi diagram.

We assign each point of the cross section to a region of the convex hull, and determine which regions need to be described more accurately. The average distance of the region points from the feature polyline is a good estimator for determining whether the region is properly described. If the region points are far from the feature polyline, we compute the Voronoi diagram for this region only. The key property of the Voronoi vertices called the largest empty circle is used to identify which region points should be identified as additional feature points. In a nutshell, we choose Voronoi vertices that are positioned on the outer side of the region points and specify which region points lie on the largest empty circle for these Voronoi vertices. These region points are identified as feature points and are added to the ordered list of the feature polyline.

In some cases, a point may lie on the largest empty circle of two or more Voronoi vertices simultaneously. Such a point can be added only once to the feature polyline and not each time it is identified as a feature point.

In the protype implementation of our method, the user may specify which regions should be processed further, and may also manually select or remove Voronoi vertices, region points or feature points if the results are erroneous (e.g. because of noisy data). Fig. 3 shows one cross section of the twisted drill bit in Fig. 2. The feature polyline in the first step is formed from the convex hull of the point set. After updating all regions of the cross section with the Voronoi method the feature polyline describes the entire cross section accurately.

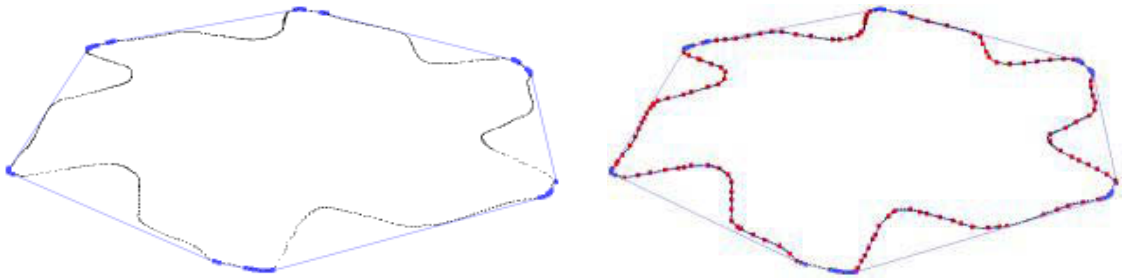

Fig. 3: The points of a cross section are processed as an individual 2D point set. (left) The convex hull of the point set (blue points) provides the initial feature polyline. (right) The feature polyline is updated as new feature points are added in regions where the point set is not accurately described (red points).

5 INSERTING NEW CROSS SECTIONS

The cross sections we have computed so far describe the point cloud accurately within a specific area. But there may be some areas between the cross sections where the point cloud is not properly described. One solution to this problem would be to divide the point cloud into more cross sections initially. But this would mean to start the whole process from scratch. Instead we have included an option in our system that allows the user to define a new intermediate cross section between two already processed cross sections, using only the feature polyline of the two cross sections.

Again, we make use of the largest empty circle, and use the Voronoi vertices as feature points of a new intermediate cross section. To be precise, not all Voronoi vertices would qualify as a feature point for the intermediate cross section, so we do not need to compute the entire Voronoi diagram of the two feature polylines. We only need the Voronoi vertices given from neighboring feature points, two from the one cross section, and one from the other. The center of the circle that is defined from these three feature points is the Voronoi vertex we identify as feature point for the intermediate cross section.

The process of matching the feature points of the two cross sections requires the triangulation of the contours defined by the two cross sections. The correspondence between vertices of the two cross sections is represented in a graph like the one described in [17]. A minimum cost path is computed along the graph, for which the vertices of the two cross sections satisfy the cost function. In our

experiments, we used the minimum distance between the points of the two cross sections as the cost function. Other approaches use different cost functions, e.g. to maximize the volume of the polyhedron that is formed by the triangle strip [10], or to minimize the surface area [7].

In our approach, we compute the Voronoi vertex defined from the first two feature points on the one cross section and the first feature point on the other cross section. Then we compute the Voronoi vertex defined from the last feature points on both cross sections. In the next step we compute the Voronoi vertex from the feature points in the middle of the cross sections, and carry on with the left and right part using a divide and conquer technique. At the end, all Voronoi vertices for all feature points on both cross sections have been computed.

The Voronoi vertices we have computed 3D vertices, so we need to project them to the corresponding plane of the new cross section. It should be noted at this point that the feature points of the new cross section are not points of the initial point cloud. They have been artificially computed to fix possible faults in the description of the point cloud from existing cross sections. Other information, such as cloud points or the convex hull is not available for this intermediate cross section, as the feature polyline is computed by other means. However, there is no problem with that, since we only use the feature polyline for further processing. Fig. 4 shows a new cross section defined by two adjacent cross sections. This part shows a detail from the nose of the Cycladic idol.

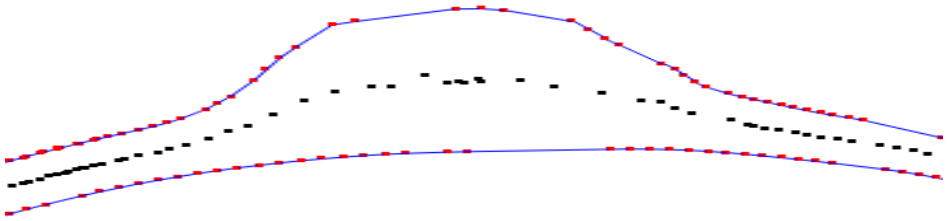

Fig. 4: The feature points that form the new cross section. Each of these points is the center of the circle defined by the three closest feature points of the two existing cross sections.

6 REPRESENTING CROSS SECTION CONTOURS BY G^1 SPLINES

The feature polylines we have computed for the cross sections of the point cloud are organized point lists and may be used to reconstruct the surface of the model. But this surface will not be smooth, as the feature polylines consist of line segments, and there may be parts where the gradient of neighboring line segments differs considerably. A possible cause for such behavior of the method could be the small number of cross sections. If the user choses increased thickness for each cross section, the parts of the point cloud will described with lower detail and the resulting model will be of lower quality. Apart from that, any possible errors that were present on the initial point cloud will also be present on the feature polylines and on the final model.

We need to represent each cross section with a smooth curve instead of a polyline. As discussed in [11], it is a common practice to use B-Spline curves, because they are easy to compute and capable of representing adequately most 3D objects. So we compute a cubic B-Spline that interpolates the points of the feature polyline. The use of a cubic B-Spline ensures that we have G^1 continuity (instead of G^0 with the polyline). We employ curves of degree 3, because it is the lowest degree satisfying G^1 continuity. The knot vector, the parameter values and the control points of the interpolating curve are determined according to [12].

Since the boundary of the cross section forms a closed point set and the feature polyline form a closed polygon, we need to keep in mind that the curve should be smooth at all points, including the beginning and the end. To ensure that the starting point / ending point satisfy G^1 continuity as the rest of the curve, we enforce a restriction that the tangent is the same in the start point and at the end point. A simple way to achieve this is to artificially include two points in the feature polyline, one

before the start/end point and one after that, so that we have three collinear points, with the middle point (being here the start/end point) having the same tangent in both directions.

There may be cases where some points of a feature polyline are positioned very close to each other. And when a B-spline tries to interpolate many data points in a limited area, it sometimes passes through the data points in an unpredictable path. This is called over fitting of the curve, and a simple way to avoid it is to use fewer knots for the interpolating curve. But this means that we should use fewer data points for interpolation. Our method allows the user to thin out the feature polyline by retaining several representatives of the feature points and discard the rest of the feature points within a specific area. After thinning the feature polyline, the interpolating B-spline should be properly computed. The user is also allowed to omit a specific feature point from the calculations if it would corrupt the resulting curve.

The process of interpolating the feature polyline with B-Splines is performed in all cross sections, and at the end, the point cloud is described by a sequence of B-Spline contours. An example of the resulting curves is shown on Fig. 4(a).

7 RECONSTRUCTING THE SURFACE

Each cross section has now been described with the use of complex structures which allow for high level processing. But the information extracted so far represents individual parts of the point cloud and there is no correlation between the cross sections. To reconstruct the surface of the object we need to combine the curves we have computed in the previous step, i.e. to define the way each cross section is associated with the previous and the next cross section.

A common way to construct a surface is to combine curves that follow different directions. In our case the issue is to select one point from each curve (i.e. for all cross sections) and to define a curve that passes through these points. That is, if we consider the cross section to be aligned horizontal, we form vertical curves that begin in the first cross section and end at the last cross section, passing through all cross sections at a specific point. The issue now is to select the proper point in each cross section to form the vertical curve. As we divided the point cloud into cross sections in the first step of our method, we now divide the curves of each cross section into curve segments. The points that define these segments will be used to form the structural elements of the surface. For example, if we want to use quads for the representation of the surface, we can use two points from one cross section and another two points from the next cross section to form a quad. If we want to use triangles, we may use the same four points and form two triangles instead of one quad.

The detail level of the resulting surface depends on the number of segments in which the user has divided the curves of the cross sections. Similar to the step where we sliced the point cloud into cross sections, if the surface of the object is smooth, the segments do not need to be very small, whereas for objects with a rough surface the segments should be small, i.e. the curves should be divided into more segments of decreased length. To increase the detail level in the resulting model we may increase the number of curve segments. The curve segments of a cross section must be of equal arc length. To calculate curve segments of equal arc length, we use the arc length parameterization [18], and choose points at proportional intervals of equal arc length on the curve. When calculating curve segments on curves with increased arc length, the curve segments will also have increased arc length compared to curve segments from smaller curves. This is normal, as we want to describe all parts of the point cloud with these curves.

At the end of this step of our method, the curves of all cross sections are divided into the same number of curve segments, and the points of these segments form a mesh that describes the surface according to the detail level specified by the user. The final model offers the user the ability to perform high level modifications to the object. Fig. 5 summarizes the final steps of the reconstruction method. Fig. 5(a) shows the B-Spline curves that describe the cross sections of the point cloud. In this example, the point cloud was divided in 100 cross sections, so 100 B-Splines describe the object. Fig.5(b) shows the vertical curves, as calculated for the 100 cross sections of the first image. 128 vertical curves have been computed for this example. Fig. 5(c) shows the B-Splines and the vertical

curves combined together. For visualization purposes only a 20% of the actual curves are rendered. Fig. 5(d) shows the reconstructed surface of the model.

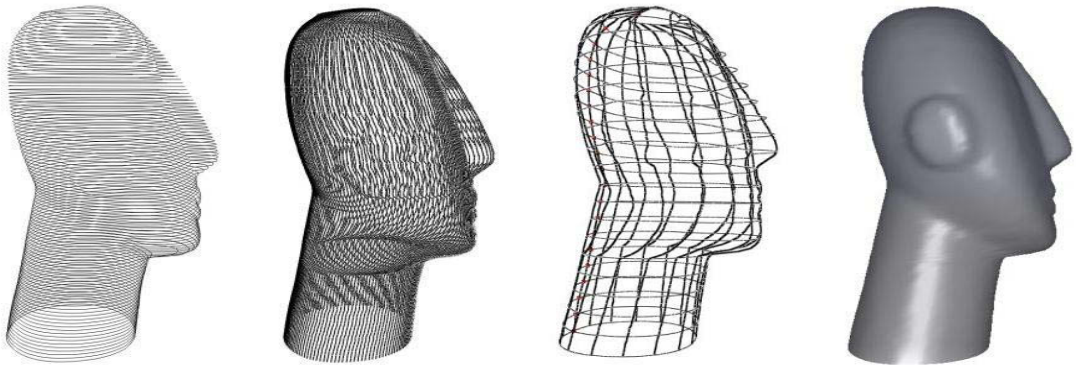

Fig. 5: The resulting model: (a) the B-Spline curves of the cross sections, (b) the vertical curves, (c) a thinned visualization of the cross sections combined with the vertical curves, and (d) the reconstructed surface.

8 EDITING

Now that we have an editable representation of the point cloud at our disposal, we may perform high level modifications on the model. If a point cloud represents an object with features such as geometric primitives (e.g. spherical or cylindrical holes or extrusions), the modifications could be directed according to the properties of these geometric primitives (e.g. modify radius, height or width of a feature). However, this is not the case with point clouds of free-form objects where no geometric primitives are present on the model. Free-form objects usually require free-form modifications, i.e. modifications that apply to the entire model or certain parts of it, but there are no properties of geometric primitives that could be used for our modifications.

In our implementation, we currently deal with free form transformations that are applied on a specific number of cross sections. The user has the ability to select one or more cross sections and apply a transformation in the form of a homogeneous transformation matrix. Rotations, translations, scaling are simply performed by multiplying each vertex of the cross section with the corresponding transformation matrix. In our implementation, the transformation matrix is built into the data structure of the cross section, so each cross section has its own transformation matrix and all elements of the cross section (cloud points, feature points, control points of the curve, Voronoi vertices etc) are subject to it.

Since the transformation matrix is a property of a cross section, we can perform modifications to at least one cross section. But as we mentioned in section 3, one or more cross sections may be selected by the user. When applying modifications, we may apply a given transformation either on the current cross section, or all the selected cross sections. The user may select as many cross sections as he or she wants, and the selected cross sections do not even have to be adjacent. For example, on a point cloud with 30 cross sections, we may select cross sections from 10 to 30, and then deselect the cross sections from 15 to 25. Then we will have cross sections from 10 to 15 and from 25 to 30 selected. The transformation will be applied to those cross sections only. Of course, if all cross sections are selected, the transformation will be applied uniformly to the entire point cloud, and if no cross sections are selected the transformation will not be applied (or will be applied to the current cross section if the user has this option enabled).

Another option allows the user either to apply a transformation on all selected cross sections evenly, or to apply a transformation proportional to the selected cross sections. For example, if we have 10 cross sections selected, and apply a scaling transformation of a factor 2x on them, we may choose to double the size of the 10 cross sections, or we may choose to scale the first cross section at

1/10 of the applied transformation, the second cross section at 2/10 and so on, until we reach the 10th cross section in which we apply the 10/10 of the transformation.

In our implementation we have even included an option to apply a mapping function that provides a function of the proportional transformation to each cross section as an individual item of the list of cross sections. This mapping function is evaluated for each selected cross section separately. The final transformation depends on the position of the cross section inside the list of selected cross sections, the mapping function, and the applied transformation as well.

This technique produces remarkable results, as the final model takes an impressive form, with just a few modifications. It is efficient especially when working on free from objects where there are no geometric primitive features present. It is up to the imagination and creativity of the user to apply certain transformations and create a model that satisfies his or her demands.

9 IMPLEMENTATION AND EXAMPLES

In this section we present examples of our application. Our method has been implemented and tested under the Microsoft Visual C++ programming environment using the Qt UI framework by Nokia. The computational geometry calculations were performed using the Qhull library [4]. Evaluations of the mapping function were made with the simple arithmetic expression evaluator developed by Robert B. Stout in the eval.c snippet.

Fig.6 illustrates parts of the UI that have been developed for testing our method. Some examples of complex transformations applied to the entire point cloud of the cycladic are illustrated in Fig. 7. Fig. 8 shows transformations applied to certain parts of the same model. Fig. 9 shows a realistic example of transformations applied to the drill bit model, which result in some variants of the original object that could be applied in practical applications.

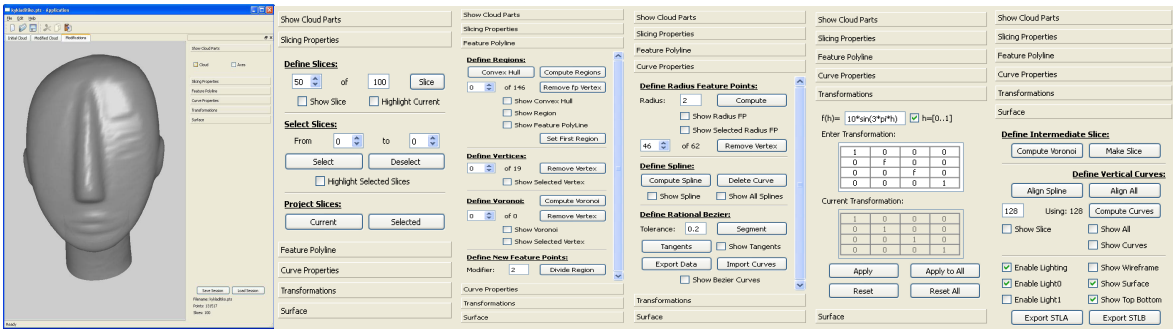

Fig. 6: The User Interface of our implementation. From left to right: (a) The entire window, with save–load options, (b) Cross section properties, (c) Feature polyline properties, (d) B-spline curve properties, (e) Transformation properties, and (f) Surface reconstruction properties.

The following tables provide details on how the resulting objects were produced. All figures represent variations of the Cycladic idol with 100 cross sections, numbered from 0 to 99. The mapping function $f(h)$ is applied to each of the selected cross sections proportional to the position of each cross section in the list of selected cross sections. For example, in the transformations of Fig. 6 we have all cross sections selected and $h = 0/99 = 0$ for cross section 0, $h = 1/99$ for cross section 1, $h = 2/99$ for cross section 2, … , $h = 98/99$ for cross section 98 and $h = 99/99 = 1$ for the last cross section. The transformation is applied on each cross section by multiplying the elements of the cross section with the appropriate transformation matrix. In the following examples we will have

Scale x y: $\begin{bmatrix} f(h) & 0 & 0 & 0 \\ 0 & f(h) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, Scale z: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f(h) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and Translate x: $\begin{bmatrix} 1 & 0 & 0 & f(h) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

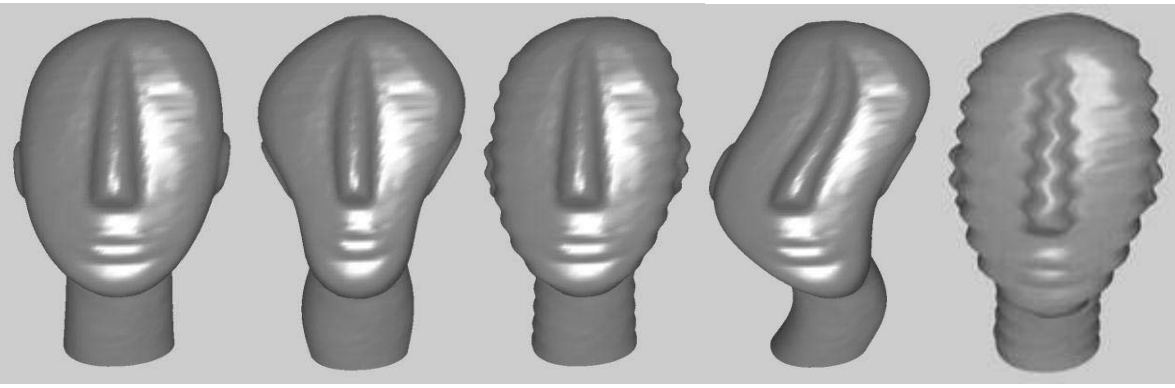

Fig. 7: Transformations applied to the entire object, (a) plain object, without any transformation, (b) large wave scaling, (c) small wave scaling, (d) large wave translation, and (e) small wave translation.

Figure	*Selected Cross Sections*	*Mapping Function*	*Applied Transformation*
Fig. 7(a)	-	-	-
Fig. 7(b)	[0..99]	$f(h)=\frac{5+\sin(10h)}{6}$	Scale x y
Fig. 7(c)	[0..99]	$f(h)=\frac{50+\sin(100h)}{51}$	Scale x y
Fig. 7(d)	[0..99]	$f(h)=10\sin(3h\pi)$	Translate x
Fig. 7(e)	[0..99]	$f(h)=1.5\sin(30h\pi)$	Translate x

Tab. 1: Details for transformations on Fig. 7.

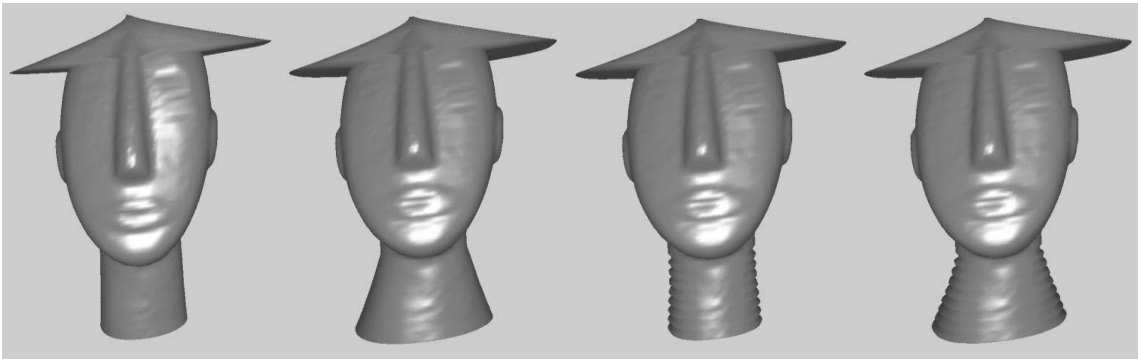

Fig. 8: Transformations applied to parts of the object, (a) 'Chinese hat', (b) 'Chinese hat' + 'growing neck', (c) 'Chinese hat' + 'wavy neck', and (d) 'Chinese hat' + 'growing neck' + 'wavy neck'.

Figure	*Selected Cross Sections*	*Mapping Function*	*Applied Transformation*
Fig. 8(a)	[85..99]	$f(h) = 2 - 2h$	Scale x y
Fig. 8(b)	[85..99]	$f(h) = 2 - 2h$	Scale x y
	[0..25]	$f(h) = 1.5 - 0.5h$	Scale x y
Fig. 8(c)	[85..99]	$f(h) = 2 - 2h$	Scale x y
	[0..25]	$f(h) = \frac{20 + \sin(50h)}{21}$	Scale x y
Fig. 8(d)	[85..99]	$f(h) = 2 - 2h$	Scale x y
	[0..25]	$f(h) = 1.5 - 0.5h$	Scale x y
	[0..25]	$f(h) = \frac{20 + \sin(50h)}{21}$	Scale x y

Tab. 2: Details for transformations on Fig. 8.

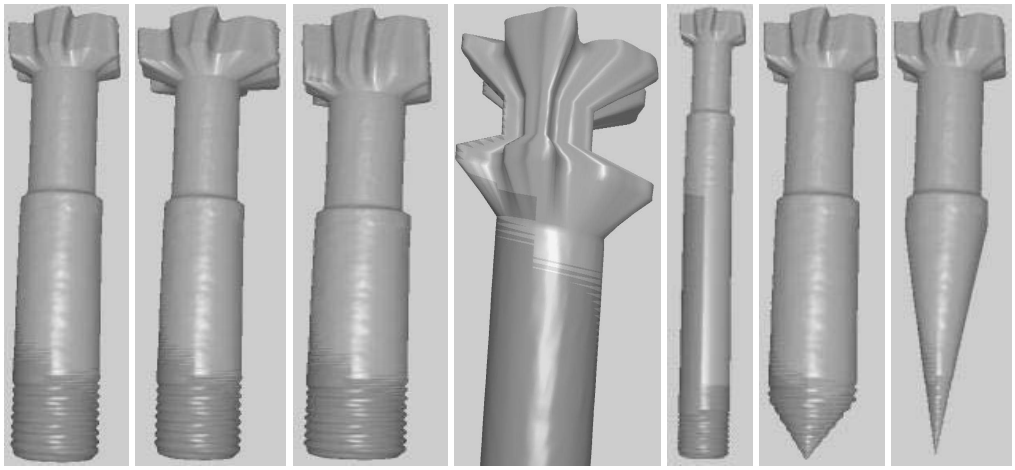

Fig. 9: Transformations applied on the twisted drill bit point cloud.

Figure	*Selected Cross Sections*	*Mapping Function*	*Applied Transformation*
Fig. 9(a)	-	-	None (Original Object)
Fig. 9(b)	[175..199]	$f(h) = 1.5$	Scale x y
Fig. 9(c)	[175..199]	$f(h) = 1.5$	Scale z
Fig. 9(d)	[178..184]	$f(h) = 1 - 0.5h$	Scale x y
	[185..190]	$f(h) = 0.5$	Scale x y
	[191..196]	$f(h) = 0.5 + 0.5h$	Scale x y
Fig. 9(e)	[0..50]	$f(h) = -50$	Translate z
Fig. 9(f)	[0..25]	$f(h) = h$	Scale x y
Fig. 9(g)	[0..100]	$f(h) = h$	Scale x y

Tab. 3: Details for transformations on Fig. 9.

10 CONCLUSIONS AND FUTURE WORK

We have introduced a novel paradigm for reconstructing the surface of an object from its point cloud, using cross sections and computational geometry algorithms. We have provided an editable representation of the object using closed cubic B-Spline curves to obtain smooth results with G^1 continuity. We have computed a point set on these curves and combined them to form vertical contours that describe the relation between cross sections for reconstructing the surface of the model. The editability of the resulting model is achieved in the form of global or partial transformations on the slices that make up our model. The properties of these transformations allow the user to produce impressive results from free-form objects. For future work, it would be interesting is to support new features that will allow for different types of editing operations, e.g. operations applied to the vertical curves we have computed on the final step of our method. One could, for instance, edit a number of vertical curves and modify a specific feature of the model.

REFERENCES

[1] Amenta, N.; Bern, M.; Kamvysselis, M.: A New Voronoi-Based Surface Reconstruction Algorithm, Computer Graphics, 32, 1998, 415–421.

[2] Au, C.K.; Yuen, M.M.F.: Feature-based reverse engineering of mannequin for garment design, Computer-Aided Design, 31, 1999, 751–759, DOI: 10.1016/S0010-4485(99)00068-8

[3] Barbero, B.-R.; Ureta, E.-S.: Comparative study of different digitization techniques and their accuracy, Comput. Aided Des. 43(2), 2011, 188-206. DOI: 10.1016/j.cad.2010.11.005

[4] Barber, C.-B.; Dobkin, D.-P.; Huhdanpaa, H.-T.: The Quickhull algorithm for convex hulls, ACM Trans. on Mathematical Software, 22(4), 1996, 469-483. DOI: 10.1145/235815.235821

[5] Bochkanov, S.; Bystritsky, V.: ALGLIB, www.alglib.net

[6] Chan, V.-H.; Bradley, C.-H.; Vickers, G.-W.: Automating laser scanning of 3D surfaces for reverse engineering, Proc. SPIE 3204, 1997, 156. DOI: 10.1117/12.294454

[7] Fuchs, H.; Kedem, Z.M.; Uselton, S.P.: Optimal surface reconstruction from planar contours, Communications of the ACM, 20(10), 1977, 693–702, DOI: 10.1145/359842.359846

[8] Geng, J.; Zhuang, P.; May, P.; Yi, S.; Tunnell, D.: 3D FaceCam: a fast and accurate 3D facial imaging device for biometrics applications, Proc. of SPIE, 5404, 2004, 316–327, DOI: 10.1117/12.542208.

[9] Jeong, W.-K.; Kahler, K.; Haber, J.; Seidel, H.-P.: Automatic generation of subdivision surface head models from point cloud data, In Proceedings Graphics Interface, 2002, 181–188.

[10] Keppel, E.: Approximating complex surface by triangulation of contour lines. IBM Journal of Research and Development, 19, 1975, 2–11, DOI: 10.1147/rd.191.0002

[11] Kyriazis, I.; Fudos, I.; Palios, L: Extracting CAD features from point cloud cross-sections, In Proc. 17-th Int. Conf. on Comp. Graphics, Visualization and Comp. Vision, Eurographics, 2009, 137-144.

[12] Lee, K.: Principles of CAD/CAM/CAE Systems, Prentice Hall, 1999.

[13] Matiukas, V.: A Survey on methods for reconsdtructing surfaces from unorganized point sets, Science – Future of Lithuania, 3(1), 2011, 10-14. DOI: 10.3846/mla.2011.002

[14] Ohtake, Y.; Belyaev, A.; Alexa, M.; Turk, G.; Seidel, H.-P.: Multi-level partition of unity implicits. ACM Trans. Graph., 22(3), 2003, 463–470, DOI: 10.1145/882262.882293

[15] Peir, J.; Formaggia, L.; Gazzola, M.; Radaelli, A.; Rigamonti, V.: Shape reconstruction from medical images and quality mesh generation via implicit surfaces, Int. J. Numer. Meth. Fluids, 53, 2007, 1339–1360. DOI: 10.1002/fld.1362

[16] Peng, X.; Zhang, Z.; Tiziani, H.-J.: 3-D imaging and modeling - Part I: acquisition and registration, Optik - International Journal for Light and Electron Optics 113(10), 2002, 448–452.

[17] Sederberg, T.W.; Greenwood, E.: A physically based approach to 2-D shape blending, ACM SIGGRAPH Computer Graphics, 26(2), 1992, 25-34. DOI: 10.1145/142920.134001

[18] Wang, H.; Kearney, J.; Atkinson, K.: Arc-length parameterized spline curves for real-time simulation, Proc. 5th International Conference on Curves and Surfaces, 2002, 387—396.

[19] Stamati V.; Fudos, I.: On reconstructing 3d feature boundaries, Computer Aided Design and Applications 5(1-4), 2008, 316–324. PMid:9368119

[20] Weng, N.; Yang, Y.-H.; Pierson, R.: Three-dimensional surface reconstruction using optical flow for medical imaging, IEEE Trans. on Medical Imaging 16(5),1997,630-641.DOI: 10.1109/42.640754