Algorithms and Tools for Deriving Editable Models from Cross-Sectional Data Sets

A Dissertation

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Ioannis Kyriazis

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

University of Ioannina

September 2017

Advisory Committee:

- **Ιωάννης Φούντος**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Θεοχάρης Θεοχάρης**, Καθηγητής, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
- **Νικόλαος Σαπίδης**, Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Δυτικής Μακεδονίας

Examining Committee:

- **Ιωάννης Φούντος**, Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Θεοχάρης Θεοχάρης**, Καθηγητής, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
- **Νικόλαος Σαπίδης**, Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών, Πανεπιστήμιο Δυτικής Μακεδονίας
- **Βασίλειος Δημακόπουλος**, Αναπλ. Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Χριστόφορος Νίκου**, Αναπλ. Καθηγητής, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
- **Γεώργιος Παπαϊωάννου**, Επίκ. Καθηγητής, Τμήμα Πληροφορικής, Οικονομικό Πανεπιστήμιο Αθηνών
- **Ιωάννης Πρατικάκης**, Αναπλ. Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Dedication

First, I would like to dedicate this work in memory of my father Thanassis, who always believed that the education of a person does never stop, but goes on for their entire life.

This work is also dedicated to my sons, Thanassis and Dimitris.

There are some special moments in life, when a problem is puzzling your mind, and the solution comes to you quite unexpectedly. Sometimes it might be a mathematical expression, or an algorithmic procedure that requires to be represented as a code snippet. Other times it may be a simple issue, such as connecting the pieces of a puzzle together, or solving a difficult level on a computer game.

In my case, the solution to such problems often popped into my mind on the least expected moments (while having lunch, in the shower, even in my sleep). Certainly, such moments offer joy and satisfaction, and therefore they are rather memorable.

Thanassi and Dimitri, I hope your life will be full of such moments.

Acknowledgements

I would like to express my sincere thanks and gratitude to my advisor, Prof. Ioannis Fudos, who has guided me through my research with helpful suggestions, good advice, encouragement and patience. Collaborating with him has been a pleasant and memorable experience.

I would also like to thank the members of my committee Prof. Theoharis and Prof. Sapidis for their suggestions and insight.

Many thanks also go to Prof. Leonidas Palios for his valuable suggestions and remarks concerning earlier stages of this research.

Last but not least, I thank my wife and colleague Katerina Soulidou, who has helped me improve my writing skills, as my co-author on several research articles concerning teaching methods on computer science addressed to primary education.

Table of Contents

List of Figures iv

List of Tables xii

List of Algorithms xiii

Abstract xiv

Εκτεταμένη Περίληψη xvi

1 Reverse Engineering - Digital Reconstruction 1

1.1 Introduction 1
1.2 Aim of the study 2
1.3 Overview 3
1.4 Contributions 4
1.5 Thesis Structure 5

2 Related Work 6

2.1 Introduction 6
2.2 Slicing and Segmentation 10
2.3 Feature Poly-lines 11
2.4 Surface Reconstruction 13
2.5 Editing 20
2.6 Centerlines 22
2.7 Comparison with our approach 25

3 Segmentation of the Point Cloud 27

3.1 About Point Clouds 27
3.2 Segmentation of the Point Cloud into Cross Sectional Slices 29

3.2.1 The Slicing Direction . . . 31
3.2.2 Thickness of the Cross Sections . . . 32
3.2.3 Empty Space between Cross Sections . . . 34
3.2.4 Other Cross Sectional Options . . . 35
3.3 Treating Cross Sections as 2D Point Sets . . . 35
3.4 Examples . . . 36

4 Computation of the Feature Poly-Line 43
4.1 Introduction . . . 43
4.2 Computation of the Feature Poly-Line . . . 44
4.3 Computation of the Convex Hull . . . 44
4.4 Ensuring proper region assignments . . . 47
4.5 Computation of the Voronoi Diagram . . . 48
4.6 Examples . . . 55
4.7 Performance Analysis . . . 59

5 Reconstruction of the Surface 61
5.1 Introduction . . . 61
5.2 Inserting new Cross-Sections . . . 62
5.3 Representing Cross-Section Contours by G^1 Splines . . . 71
5.4 Reconstructing the Surface . . . 76

6 Editing Free Form Models 86
6.1 Introduction . . . 86
6.2 Applying Free Form Transformations . . . 87
6.3 Implementation and Examples . . . 88

7 Editing Operators for Cross-Sectional Data-Sets 99
7.1 Introduction . . . 99
7.2 Properties of the Input Model . . . 100
7.3 Editing Operators . . . 102
7.3.1 The Curve of Centroids . . . 102
7.3.2 Alignment Operator . . . 105
7.3.3 Displacement Operator . . . 106
7.3.4 Elastic Operator . . . 109

7.3.5 Inflation Operator . . . 111
7.3.6 Placing the curve of centroids on a parametric function . . . 113
7.3.7 Cross-Sectional Free-Form Editing . . . 115
7.3.8 Limitations . . . 116
7.4 Implementation and Examples . . . 117

8 Conclusions - Future Research Directions **122**
8.1 Validation - Comparison to Commercial Tools . . . 122
8.2 Conclusions . . . 127
8.3 Future Research Directions . . . 128

Bibliography **129**

A Rejected Approaches **147**
A.1 Introduction . . . 147
A.2 The Correlation Coefficient . . . 148
A.2.1 Reason for Rejecting this Approach . . . 154
A.3 The Alternating Convex Hull . . . 155
A.3.1 The Internal - External Point switching effect . . . 157
A.4 The Voronoi diagram based approach - Variations . . . 159
A.4.1 Using the farthest Voronoi Vertex . . . 159
A.4.2 Using a single Voronoi Vertex from a bounded area . . . 161
A.4.3 Using multiple Voronoi Vertices . . . 163
A.5 The Rational Bezier Curve Fitting Approach . . . 165
A.5.1 Reason for Rejecting this Approach . . . 171

List of Figures

1.1 Overview of our method. 4

3.1 The point cloud of a female figure. The only information provided are the $[x, y, z]$ coordinates of points on the surface of the model. 28

3.2 (a) The point cloud of a salt dome with one cross section highlighted. (b) The point cloud is divided into several cross sections. (c) Top view of the cross sections. 30

3.3 The slicing direction is an essential parameter for the modeling process. (a) The slicing direction matches the cylinder axis. The points of a cross section form a circle. (b) The slicing direction is different than the cylinder axis. The points of a cross section form an ellipse. 31

3.3 The slicing direction is an essential parameter for the modeling process. (c) The slicing direction is vertical to the cylinder axis. The points of a cross section form a rectangle. (d) The slicing direction is vertical to the cylinder axis. The points of the last cross section form a line. 32

3.4 The thickness of the slices is another essential parameter. (a) Each slice is required to provide the proper information about the boundary of the model in its context. (b) Very thick slices may result in having features tangled together. (c) A proper slice thickness may eliminate problems caused by an inappropriate slicing direction. (d) Very thin slices may fail to describe a feature due to lack of information. 33

3.5 The Cyberware Screwdriver. The point cloud may be sliced into cross sections of variable thickness. The principal axis of the point cloud usually matches the ideal direction for slicing the cloud. 37

3.6 Instances of the sliced screwdriver with different values of thickness. When the highlighted cross section from (a) is projected on its 2D plane, we get the slice in (b). Having cross sections with half its thickness will project to slice (c). 38
3.6 Instances of the sliced screwdriver with different values of thickness. A cross section with $1/4$ thickness projects to slice (d). The entire screwdriver sliced with a specific thickness (e), and half its thickness (f). . . 38
3.7 Detail of the Cyberware Screwdriver. (a) One cross section is highlighted, and (b) the points of the cross section are projected to its corresponding 2D plane. 39
3.8 The Cyberware hip bone. The proper slicing direction is not obvious in this model, as the medial axis of the cloud is not a simple curve. . . 40
3.9 The Cycladic Idol. (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. 41
3.10 A twist drill bit. (Up) The actual model, (Down) The point cloud sliced into cross sections. 42

4.1 A point set with the shape of a rectangle is convex. The feature poly-line matches the ordered point list defined by the convex hull, as it describes the boundary sufficiently at all regions. 45
4.2 A cross section of the cycladean idol point cloud. The vertices of the convex hull are identified as feature points for this cross section. Some regions are not yet described by the feature poly-line. 46
4.3 A cross section of the point cloud of the Cyberware boat [1]. 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 need to use the information of their neighboring points to assign them to the correct region. 47
4.4 Detail of a cross section of the handle of the screwdriver point cloud. The largest empty circle of a Voronoi vertex is used to identify additional feature points and update the curve. 50

4.5 The user requests the number of centers to be used for identifying additional feature points. The centers closest to the reference points defined at equal intervals on the outside of the feature poly-line are used. 52
4.6 Detail of a cross section of the cycladean idol point cloud. (a) The convex hull provides the initial feature poly-line. (b) Some regions are not properly described. (c) The Voronoi diagram is computed for those regions. (d) Additional feature points are identified. (e) The feature poly-line is updated. 55
4.7 A cross section of the cycladean idol point cloud. For the feature poly-line to fully describe a region, several iterations may be required. (Top) The first iteration of our method applied on a region. (Bottom) The second iteration for the small region. 56
4.8 A cross section of the cycladean idol point cloud (cont.). After the second iteration the region is described as expected. All other regions have also been processed. 57
4.9 A cross section of the twist drill bit point cloud. The points of a cross section are processed as an individual 2D point set. (Top) The convex hull of the point set (blue points) provides the initial feature poly-line. (Bottom) The feature poly-line is updated as new feature points are added in regions where the point set is not accurately described (red points). 58

5.1 In case a new intermediate cross section is required, it is constructed from the Voronoi centers of the points of two adjacent feature poly-lines. 63
5.2 For the contour triangulation, the connectivity of the points is expressed as a monotonically decreasing path in a tabular scheme that specifies which points participate in each triangle or quad. 64
5.3 The recursive process of the contour triangulation. (a) The points p_i of the one cross section are positioned as a point list on axis x, while the points q_i of the other cross section are arranged as an ordered point list on axis y. The process includes: (b) choosing the middle point on the cross section with the greater number of points, ... 67

5.3 ... (c) finding its closest neighbor on the other cross section, (d) defining a correspondence between the two points, and (e, f, ...) repeating for the point lists on the left and right side recursively. 68
5.3 The recursive process of the contour triangulation (cont.). 69
5.3 The recursive process of the contour triangulation (cont.). 70
5.4 Cross sections in the area under the chin of the cycladean idol point cloud. (Left) A triangulation is computed between the contours of the two cross sections. The three vertices of each triangle correspond to a Voronoi region and define a circle (the largest empty circle), the center of which is identified as feature point for the new intermediate cross section. (Right) A detail from the area under the chin of the cycladean idol. The newly computed feature points are positioned in 3D space and have to be projected to the corresponding 2D plane between the two existing cross sections, to form the feature poly-line of the intermediate cross section. 71
5.5 To ensure G^1 continuity on the end points we insert two more points that enforce collinearity of the three points near the start/end point of the curve. 74
5.6 A detail of a smooth curve. (Up) If the feature points are very close to each other, the resulting curve may have irregular or even self intersecting sections due to over-fitting. (Down) Discarding selected points to thin out the feature poly-line will correct the problem of over fitting and produce accurate curves. 75
5.7 The sequence of curves of the entire cycladean idol point cloud (100 cross sections). 76
5.8 Interpolating a curve on the existing feature points does not produce good results, when the points are not properly aligned. (Left) The correspondence between the feature points. (Right) The surface of the resulting model. 77

5.9 (Left) The start/end point p_i on each cross section is set to the feature point with $p_x = min(x)$ and $p_y = min(|y|)$. The points are not aligned and the surface reconstruction will suffer from irregularities. (Right) By re-sampling the curves we are able to set the start/end points p_i to the points with $p_x = min(x)$ and $p_y = 0$. The points are now aligned and the surface reconstruction becomes robust. 79
5.10 Interpolating vertical B-Splines on the newly re-sampled feature points according to the arc-length method ensures a proper representation of the model surface with a one-to-one correspondence between the points across all cross sections. 84
5.11 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. . 85

6.1 The User Interface of our implementation. From left to right: (a) The entire window, with save-load options, (b) Cross section operations, (c) Feature poly-line properties, (d) B-spline curve operations, (e) Transformation properties, and (f) Surface reconstruction properties. 89
6.2 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. 91
6.3 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'. 93
6.4 Transformations applied to parts of the object, (e) 'Asian Monk hat', (f) 'Gandalf the grey'. 95
6.5 Transformations applied on the twist drill bit point cloud. 97

7.1 The model of an artery, and the points that came as input. **Input**: A 3D point set of an artery, organized in 2D slices of 80 points, provided from a CT Scan. **Output**: A collection of closed NURBS curves that interpolate the points of each slice accurately. These curves are used for reconstructing the surface of the model (red surface on the top left). 101

7.2 (Front) The curve that interpolates the centroids of the slices can be edited to give the model new features (Red: initial curve - Blue: edited curve). (Back) The surface of the model is adjusted according to the modification of the curve of centroids. 103
7.3 Alignment of the selected cross sections to the curve of centroids. . . . 105
7.4 Translation using the displacement operator. (a) The initial model in red, (b) the edited model in blue, (c) details of the operator. 107
7.5 Rotations using the displacement operator. 109
7.6 The centroid C_i is getting closer to the line segment [C_1, C_2], as the parameter α increases. For $\alpha = 0$ we have $C_i' = C_i$, and for $\alpha = 100\%$ the centroid C_i is projected on the line segment [C_1, C_2]. 110
7.7 (Left) The initial model (a), the edited model (b) and the transformation (c). (Right) Applying 25%, 50%, 75% and 100% tension to the selected centroids. 111
7.8 The curve of centroids remains unaffected by the transformation, but the points of the selected cross sections are scaled around their centroids. (a) The initial model, with the selected cross sections inflated. (b) The inflated model. (c) For $s < 1$ the operator causes deflation. (d) The transformation. From centroid C_0 to C_1 and from C_2 to C_3 , scaling is applied proportionally. From C_1 to C_2 the cross sections are completely scaled. 112
7.9 Editing the cross sections with a parametric function. Each cross section is positioned on a path defined by a mathematical expression according to its index in the selected region. Here the curve of centroids is placed on a sine function. (a) Initial model, (b) edited model, (c) the transformation. 114
7.10 (Up) The user defines a set of feature points to form a helix in R^3 and interpolates a NURBS curve on these points. (Down) The curve of centroids of the model is placed on this curve, causing the model to deform in the shape of the helix. 115
7.11 The user interface. 118
7.12 The path of an artery can be redefined by editing the curve of centroids. This is a key feature for medical applications such as the bypass surgery. 119

7.13 The surgical operation of angioplasty often includes a stent insertion. When a stent is inserted, the tissue of the artery is both stretched, as described with the Elastic Operator, and also inflated, as described with the Inflation operator. 120
7.14 (a) The drill bit model without any modifications, (b) Using the displacement operator to translate parts of the model, (c) Using the inflation operator to deflate parts of the model, and (d) Using the displacement operator to rotate a part of the model around its center. 121

8.1 The initial drill bit point cloud consists of 1436231 points, but after applying our method, the resulting model consists of only 25600 points, which are ordered, in 200 slices x 128 points per slice. 122
8.2 The drill bit model edited with Pointools. Parts of the point cloud have been painted to show the different parts of the model: Screw threads (orange), body (blue), proportionally displaced cross sections (pink), displaced region (purple), drill (green). 123
8.3 The drill bit model edited with Geomagic Studio. Some triangles at the screw threads and the drill seem to have been incorrectly computed. . 124
8.4 The surface of the drill bit model, reconstructed with the ball pivoting algorithm in MeshLab. 124
8.5 Editing with 3ds Max. Editing is performed by manually deforming individual control points on the bounding box. 126

A.1 A thinning technique can be applied to the 2D point set, to construct a closed polygon curve that accurately represents the shape implied by these points. 148
A.2 Values of the correlation coefficient ρ for data sets of variable point dispersion (ρ is represented as r in the source of this figure [2]). 149
A.3 Initial point IP determination and first, second segment construction. . 150
A.4 When applied to parts of a cross section of the screwdriver point cloud, the thinning method using the correlation coefficient does not give satisfying results. For this reason, this approach has been rejected. 154

A.5 Detail of a cross section of the handle of the screwdriver point cloud. The points in the highlighted region are isolated and the convex hull of these points is computed to identify the next set of feature points for the curve. 155
A.6 A cross section of the handle of the screwdriver point cloud. (a) Second iteration of the Alternating Convex Hull Method. There are still some regions where the curve doesn't fit the points accurately. (b) Third iteration of the Alternating Convex Hull Method. The curve now describes the slice points more accurately. 156
A.7 A detail of a cross section of the handle of the screwdriver point cloud. The convex hull method identifies internal and external slice points alternatively as feature points. If this affects the curve significantly, the feature poly-line will have irregularities when switching between regions. 158
A.8 A cross section of the handle of the screwdriver point cloud. The largest empty circle of the farthest Voronoi vertex is used to identify the next group of feature points. 160
A.9 A cross section of the handle of the screwdriver point cloud. By choosing a Voronoi vertex that is the farthest inside a bounded region we can identify feature points that are not too close to each other, and update the fitting poly-line faster. 162
A.10 A cross section of the hip bone point cloud. By choosing several Voronoi vertices at once we can update the fitting poly-line even faster. 164
A.11 The unit vectors of the tangents at the end control points are estimated by expressing each tangent of each point as a linear combination of its 4 closest neighbors [3]. 167
A.12 A new rational Bezier curve is inserted each time the tangent vectors switch sides in respect to the direction of the feature poly-line. 170
A.13 The feature poly-line is segmented into regions for fitting the individual rational Bezier curves. Continuity and smoothness on the edges is ensured by setting the control points on the edges to be collinear. . . . 171
A.14 The over fitting effect is present in many parts of the model. Removing feature points that are close to each other will solve the problem. However, the number of rational Bezier fitting curves that have to be calculated again is large. 172

List of Tables

6.1 Details for transformations on Fig. 6.2. 92
6.2 Details for transformations on Fig. 6.3. 94
6.3 Details for transformations on Fig. 6.4. 96
6.4 Details for transformations on Fig. 6.5. 98

8.1 Comparison with other commercial tools (surface reconstruction). . . . 125
8.2 Comparison with other commercial tools (editing). 127

List of Algorithms

4.1 The algorithm for the feature point extraction using the Voronoi diagram. 54
5.1 The algorithm for the contour triangulation. 65
5.1 The algorithm for the contour triangulation. 66
7.1 The algorithm for the Displacement Operator. 108
7.2 The algorithm for the Elastic Operator. 110
7.3 The algorithm for the Inflation Operator. 113
7.4 The algorithm for the Parametric Function Operator. 114
A.1 The algorithm for finding the initial point in the thinning approach using the correlation coefficient. 151
A.2 The algorithm for finding the radius of the neighborhood of S^i. 152
A.2 The algorithm for finding the radius of the neighborhood of S^i. 153
A.3 The algorithm for the Alternating convex hull. 157
A.4 The algorithm for the Voronoi method, using the farthest Voronoi vertex. 161
A.5 The algorithm for the Voronoi method, using the farthest Voronoi vertex within a bounded area. 163

Abstract

Ioannis Kyriazis, Ph.D., Department of Computer Science and Engineering, University of Ioannina, Greece, September 2017.
Algorithms and Tools for Deriving Editable Models from Cross-Sectional Data Sets.
Advisor: Ioannis Fudos, Professor.

In this thesis, we present a novel method for reconstructing the surface of a 3D object using as input only 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. This model will provide a high level representation of the actual object, which can also be modified with the use of a set of editing operators we have defined. The entire method is performed in three phases: the segmentation of the point cloud, the reconstruction of the model and the editing process.

In the segmentation phase, the point cloud is sliced into cross sections, which are later on treated as 2D point sets. Several parameters of the slicing procedure influence the resulting model, and have to be set properly to ensure high quality output. One essential parameter is the slicing direction. It has to correspond to the principal axis of the point cloud, either locally or globally, so that the basic features of the point cloud are properly distributed across the cross sections. Another key parameter in the segmentation process is the thickness of the cross sections. The cross sections should contain sufficient information about the features of the point cloud, so their thickness cannot be very slim. But it should not be very thick either, as we do not want to have many features tangled together in the same cross section.

The reconstruction phase includes several procedures that take as input the cloud points of the cross sections and provide as output a model of the 3D object, with high level features and properties that can be used for further processing. First, the boundary of each cross section is represented as a set of line segments called the

feature poly-line. This poly-line is computed with the use of computational geometry methods, i.e. the convex hull and the Voronoi diagram. The feature poly-line of each cross section is then recomputed as an interpolating B-Spline curve, providing a smooth continuous representation of the model. The curves of the neighboring cross sections are combined with each other to form the surface of the final 3D model. To reconstruct the surface between two adjacent cross sections, a new set of feature points is computed on the B-Spline curves, using arc length parameterization, and a contour triangulation method provides the surface of the model. The resulting model can be subject to high level modifications that provide variations of the initial object with additional user specified properties.

For the editing process, we have implemented a set of editing operators, which can be applied either to the entire model, or on a part of it, to deform its surface in various ways. As a point of reference for these editing operators, a given point may be used on the interpolating curve computed from the centroids of the cross sections. We call it the *curve of centroids* and it is a form of skeleton curve for the model. The editing operators vary from general purpose transformations to high level editing operators addressed to cross sectional data sets used in medical research. The general purpose transformations can be used to modify free form models arbitrarily, and can be applied to parts of the model according to user specified parameters. The high level editing operators allow modifications such as bending or stretching the model of an artery, allowing medical experts to study and control the behavior of tissues during a simulation of a surgical operation such as angioplasty or stent insertion.

The novelty of this study includes:

- the introduction of an efficient method for computing a boundary representations of cross-sectional point sets, using the convex hull and the Voronoi diagram of the points,
- the use of a technique for inserting artificial cross sections between existing slices, to improve the rendering quality of the model, and
- the definition of a set of editing operators that may be used in everyday examples, varying from purely aesthetic purposes to simulation of surgical operations in medical data sets.

Εκτεταμενη Περιληψη

Ιωάννης Κυριαζής, Δ.Δ., Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Σεπτέμβριος 2017.
Αλγόριθμοι και Εργαλεία για την Παραγωγή Επεξεργάσιμων Μοντέλων από Σύνολα Δεδομένων Οργανωμένων σε Εγκάρσιες Τομές.
Επιβλέπων: Ιωάννης Φούντος, Καθηγητής.

Στη διατριβή αυτή παρουσιάζουμε μια καινοτόμα προσέγγιση για την ανακατασκευή της επιφάνειας ενός 3-διάστατου αντικειμένου, χρησιμοποιώντας ως είσοδο ένα νέφος σημείων της σαρωμένης επιφάνειάς του. Στόχος είναι να αποκτήσουμε ένα μοντέλο CAD με δυνατότητα επεξεργασίας, που να μπορεί να κατασκευαστεί και το οποίο θα περιγράφει με ακρίβεια τη δομή και την τοπολογία του νέφους σημείων. Το μοντέλο αυτό θα παρέχει μια υψηλού επιπέδου αναπαράσταση του αντικειμένου, που θα μπορεί να τροποποιηθεί με τη χρήση ενός συνόλου από τελεστές τους οποίους έχουμε σχεδιάσει. Η όλη μέθοδος διενεργείται σε τρεις φάσεις: την κατάτμηση του νέφους σημείων, την ανακατασκευή της επιφάνειας του μοντέλου και την επεξεργασία/τροποποίηση του μοντέλου.

Κατά την φάση της κατάτμησης, το νέφος σημείων χωρίζεται σε εγκάρσιες τομές, οι οποίες κατά την μετέπειτα επεξεργασία αντιμετωπίζονται ως 2-διάστατα σύνολα σημείων. Σε αυτή τη φάση υπάρχουν κάποιες παράμετροι που επηρεάζουν το τελικό μοντέλο, και θα πρέπει να καθοριστούν κατάλληλα ώστε να εξασφαλιστεί το καλύτερο δυνατό αποτέλεσμα. Μια βασική παράμετρος είναι η κατεύθυνση της κατάτμησης, που θα πρέπει να συμφωνεί με τον κύριο άξονα του νέφους σημείων, είτε σε καθολικό επίπεδο είτε τοπικά, ώστε τα βασικά χαρακτηριστικά του νέφους σημείων να αποτυπωθούν επαρκώς στις εγκάρσιες τομές. Μια άλλη βασική παράμετρος στη φάση της κατάτμησης είναι το πάχος που θα έχουν οι τομές. Οι τομές θα πρέπει να περιέχουν επαρκή πληροφορία για τα χαρακτηριστικά του νέφους

σημείων, οπότε δεν θα πρέπει να είναι υπερβολικά λεπτές. Όμως δεν θα πρέπει να είναι ούτε και πολύ παχιές, για να μην εμπλέκονται στην ίδια τομή πολλά χαρακτηριστικά μεταξύ τους ιδιαίτερα όταν αυτά εντοπίζονται σε γειτονικές περιοχές.

Η φάση της ανακατασκευής του μοντέλου περιλαμβάνει αρκετές διαδικασίες, οι οποίες παίρνουν ως είσοδο τα σημεία των εγκάρσιων τομών και παράγουν ως έξοδο ένα μοντέλο του 3-διάστατου αντικειμένου, που διαθέτει υψηλού επιπέδου ιδιότητες και χαρακτηριστικά, και μπορεί να χρησιμοποιηθεί για περαιτέρω επεξεργασία. Αρχικά τα όρια της κάθε τομής αναπαρίστανται με μια κλειστή πολυγωνική γραμμή, η οποία υπολογίζεται με τη βοήθεια μεθόδων υπολογιστικής γεωμετρίας, όπως το κυρτό περίβλημα και το διάγραμμα Voronoi. Στη συνέχεια, η πολυγωνική γραμμή κάθε τομής επαναπροσδιορίζεται ως μια καμπύλη B-Spline που θα παρέχει μια συνεχή και ομαλή αναπαράσταση του μοντέλου. Οι καμπύλες των γειτονικών τομών συνδυάζονται για να συνθέσουν την επιφάνεια του τελικού 3-διάστατου μοντέλου. Για την ανακατασκευή της επιφάνειας μεταξύ δυο γειτονικών τομών, υπολογίζουμε ένα νέο σύνολο σημείων επάνω στις καμπύλες, χρησιμοποιώντας παραμετροποίηση μήκους τόξου, και η επιφάνεια του μοντέλου προκύπτει από την τριγωνοποίηση των περιγραμμάτων. Το μοντέλο που προκύπτει μπορεί να υπόκειται σε υψηλού επιπέδου μετασχηματισμούς που παρέχουν παραλλαγές του αρχικού αντικειμένου με επιπλέον ιδιότητες καθορισμένες από το χρήστη.

Στη φάση της επεξεργασίας, υλοποιήσαμε μια σειρά από τελεστές που μπορούν να εφαρμοστούν στο μοντέλο είτε καθολικά είτε τμηματικά, για να μετασχηματίσουν την επιφάνειά του με διάφορους τρόπους. Το σημείο αναφοράς για τους μετασχηματισμούς αυτούς τοποθετείται πάνω στην καμπύλη που ορίζεται από τα κέντρα βάρους των τομών και η οποία αποτελεί ένα είδος σκελετού του αντικειμένου. Οι τελεστές ποικίλουν από μετασχηματισμούς γενικού ενδιαφέροντος μέχρι υψηλού επιπέδου μετασχηματισμούς ειδικά σχεδιασμένους για δεδομένα προερχόμενα από τομογραφίες που χρησιμοποιούνται στην ιατρική έρευνα. Οι απλοί μετασχηματισμοί χρησιμοποιούνται για την ελεύθερη τροποποίηση ενός μοντέλου, και μπορούν να εφαρμοστούν σε τμήματα του μοντέλου σύμφωνα με τις παραμέτρους που ορίζει ο χρήστης. Οι τελεστές υψηλού επιπέδου αφορούν τροποποιήσεις όπως πχ το τέντωμα ή το λύγισμα του μοντέλου μιας αρτηρίας, επιτρέποντας στους ειδικούς να μελετήσουν και να ελέγξουν τη συμπεριφορά των ιστών κατά τη διάρκεια μιας

εγχείρησης, όπως πχ στην αγγειοπλαστική ή στην τοποθέτηση stent.

Η συνεισφορά της έρευνας αυτής περιλαμβάνει:

- τη σύσταση μιας μεθόδου για τον αποτελεσματικό υπολογισμό της αναπαράστασης των ορίων ενός συνόλου σημείων οργανωμένων σε τομές,
- τη χρήση μιας καινοτόμου τεχνικής για την εισαγωγή τεχνητών τομών ανάμεσα σε υπάρχουσες τομές για τη βελτίωση αναπαράστασης του μοντέλου, και
- τον ορισμό ενός συνόλου από τελεστές μετασχηματισμού που μπορούν να εφαρμοστούν σε γενικά παραδείγματα, που ποικίλουν από αισθητικές παρεμβάσεις μέχρι την προσομοίωση χειρουργικών επεμβάσεων σε ιατρικές εφαρμογές.

Chapter 1

Reverse Engineering - Digital Reconstruction

1.1 Introduction

1.2 Aim of the study

1.3 Overview

1.4 Contributions

1.5 Thesis Structure

1.1 Introduction

Many applications in manufacturing, medicine, geography, design, and entertainment require the 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 [4, 5, 6, 7]. The process is brought down to obtaining a point cloud, i.e. a set of points that lie on the boundary of a 3D object, using one of several 3D point acquisition methods available, e.g. laser scanning, photogrammetry or CT scan [8], and then process this point cloud to extract a CAD model of the object surface. For example, the point cloud may be acquired with the use of a 3D laser scanner [9], or by identifying feature points on multi-camera images [10, 11], or even computerized tomography when it comes to medical applications [12, 13]. For processing the point cloud, various methods

have been proposed, which include slicing the point cloud into cross-sections [12], or patches [14], or treating the point cloud as a whole [15].

Editing such models can apply to many situations and for various purposes, such as CAD, where the designer may create models of tools, parts or free-form objects for industrial use. In this case, the designer creates the model from scratch and defines all the features and properties of the object they are intended to create.

Another application is Reverse Engineering, where the tool or part is already available to the designer. In this case the aim is to identify the properties or features of the object and to create CAD models that satisfy the requirements for reproducing these properties or features. The next step in this process is to define additional features and design another part or tool that has additional properties than the original object.

Another interesting example of 3D model editing is for medical purposes [12, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27], where the 3D representation of human tissues is used for simulating medical procedures, such as needle insertions in soft tissues, or stent insertion inside arteries. The deformation of the tissues in these cases is considered as editing the model of the 3D object.

Other examples of editing 3D models include educational applications [28], where the students learn the principles of programming with the help of a 3D object that reacts according to their commands, and also in gaming [29], where the player controls a 3D character for recreation. These cases may include 3D object editing in the form of deformations [30], morphing [31], free-form animations etc.

Editing of 3D models may as well be used for aesthetic purposes, i.e. to produce art works using CAD [32, 33, 34].

1.2 Aim of the study

Depending on their purpose and the means they use to achieve their goal, some works are primarily oriented towards reverse engineering, others in surface reconstruction, others in editing, etc. However, for someone who would be interested in using a tool that is capable of both

a) extracting a model from an unstructured input, and

b) to perform editing,

the available tools are extremely complex, as they require expert skills to use them. For this reason, we have developed a method that complies to the afforementioned user requirements and offers both model extraction and editing.

The aim of this work is to introduce a framework for reverse engineering objects of mechanical or free form design to obtain fully editable 3D CAD models that can be reproduced or modified before production. More specifically, our approach aims to partition a 3D point cloud into cross-sectional slices, so as to create an editable and parameterized CAD model described as a set of slices. This type of model provides the user-designer with the capability of editing, redesigning and reproducing the original object, depending on his preferences and needs, by editing either the entire set of slices, or a selected part of the model.

1.3 Overview

The framework of our approach is summarized in Figure 1.1. We begin by importing a 3D point cloud which may or may not be structured. In some cases, eg. when the point cloud is the result of a CT scan, the input is already structured, meaning that the points are ordered and grouped into cross-sectional slices. In such cases we do not have to slice the cloud, and proceed directly on the next step. Other times the point cloud will not be structured, meaning that the points are not ordered and grouped into slices. In such cases, we divide the point cloud into cross sectional slices, and process each slice as if it were a 2D point set. For this point set, we compute a linear feature poly-line, using the convex hull and the Voronoi diagram of the points. Then we construct a smooth continuous B-Spline curve that interpolates the points of the feature poly-line in each slice. After the construction of our model, in terms of a set of parallel smooth curves, we define a set of editing operators that can be applied on selected slices, to apply modifications on the model. The editing operators vary from low level free form transformations, to high level editing operators that make use of a feature called the curve of centroids, and perform modification on the model according to high level user specified parameters controlled by the user.

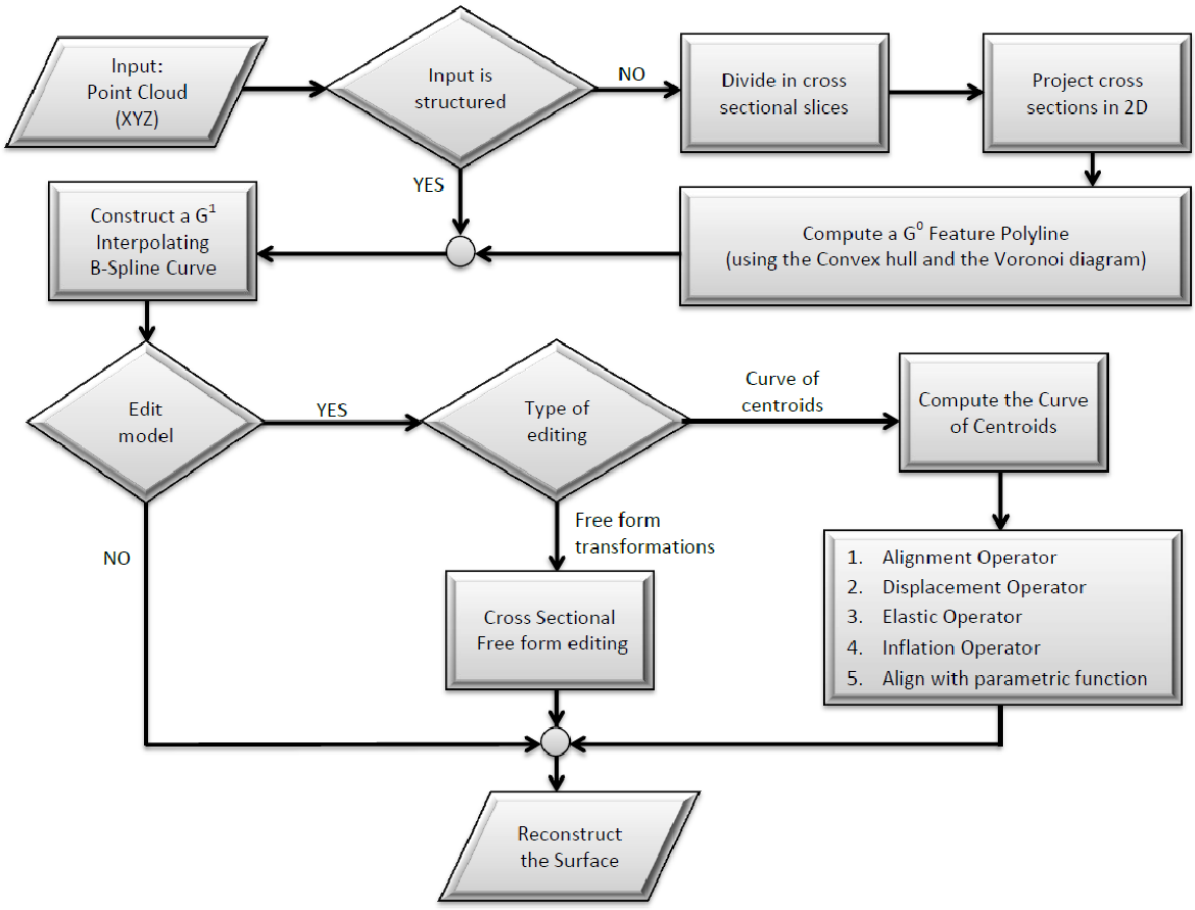


Figure 1.1: Overview of our method.

1.4 Contributions

The method presented in this thesis can be applied to point clouds of objects of mechanical or free form design. This thesis makes the following technical contributions:

- It presents a slicing method that partitions a point cloud into parallel slices that will be refined to represent the boundary of each cross-section using a subset of the points, i.e. some feature points of a slice.
- It introduces an efficient method for computing a boundary representation of the cross sectional point sets, using the convex hull and the Voronoi diagram of the points.
- It presents a method for surface reconstruction that is used for visualization purposes.

- It introduces a technique for inserting artificial cross sections between existing slices, to improve the rendering quality of the model.
- It describes a methodology for performing free form editing on the model.
- It introduces a set of editing operators that may be used in everyday examples, varying from purely aesthetic purposes to simulation of surgical operations in medical data sets.
- It validates our approach by comparing the results of our tool-set with several commonly used commercial applications.

1.5 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 presents some Related Work. Chapter 3 discusses the details on segmenting the point cloud, the interactive slicing technique and how the slicing parameters affect the subsequent processing steps. In chapter 4 we provide details on how to compute a feature poly-line for each cross section of the sliced point cloud, using methods based on the convex hull and the Voronoi-diagram of the point sets. In chapter 5 we offer details on computing planar B-Spline curves fitting the poly-line data with G^1 continuity. We also introduce a method for reconstructing the surface of the object based on the cross section B-splines. Chapter 6 presents a novel paradigm for applying local or global editing operations to the derived model. We provide implementation considerations and some examples of applying our free form editing tool. In chapter 7 we offer a set of high level editing operators addressed to cross sectional data sets used in medical research. Finally, in chapter 8 we make our conclusions, we compare the results of our method to other tools commonly used, and deploy some thoughts for future research directions. An Appendix is also available, in which we describe some approaches that have been studied, fully implemented, but eventually rejected, as they did not provide the quality results we have been expecting.

Chapter 2

Related Work

2.1 Introduction

2.2 Slicing and Segmentation

2.3 Feature Poly-lines

2.4 Surface Reconstruction

2.5 Editing

2.6 Centerlines

2.7 Comparison with our approach

2.1 Introduction

Several methods have been developed that extract features from a point cloud. Some of these methods are applied on mechanical objects, and others on free-form objects. While mechanical objects are easy to describe with a standard set of features, representing free-form objects necessitates the expansion of the usual repertoire of features and operation with innovative constraint-based features.

Yano and Harada [35] proposed a method to reconstruct B-Spline surfaces from generalized cylindrical meshes by skinning. They automated the construction of sectional curves on a triangle mesh in order to convert the mesh into a skinning surface. Their method generates a smooth scalar field on mesh by solving a Laplacian equation, then create a set of gradient flows by tracing the field from a selected point on the mesh. The gradient flows are used for quadrilateral remeshing.

Yuwen et al [36] addressed the B-spline surfaces fitting and direct slicing problem from 3D measured data. Firstly, a method of extracting cross-sectional contours from a point cloud is used, then a cross-sectional design technique with the possibility of the direct integration between reverse engineering and rapid prototyping. For a good initial parameterization, a method of constructing the base surface is provided. Based on the parameter correction strategy, an iterative B-spline surface fitting algorithm with tight tolerance is accomplished.

Wang et al [37] introduced a method for surface reconstruction from sectional contours, which is based on the consistent utilization of the two-dimensional constrained Delaunay triangulation. The triangulation is used to extract the parametric domain and to solve the problems associated with correspondence, tiling and branching in a general framework. Natural distance interpolations are performed in order to complete the mapping of the added intermediate points. Surface smoothing and remeshing are conducted to optimize the initial surface triangulations.

Daniels and Cohen [38] presented an algorithm for generating a smooth surface between two closed spatial spline curves that generates a temporal deformation between the input curves, which can be thought of as sweeping a surface. Their method generates intermediate curves that behave like wavefronts as they evolve from the shape of one boundary curve to a medial type curve, and then gradually take on the characteristics of the second boundary curve.

Chai et al [39] used a gradient controlled partial differential equation surface to express terrain surfaces, in which the surface shapes can be globally determined by the contours, their locations, height and gradient values. The surface generated by this method is accurate in the sense of exactly coinciding with the original contours and is smooth with C^1 continuity everywhere. The method can reveal smooth saddle shapes caused by surface branching of one to more and can make rational interpolated sub-contours between two or more neighboring contours.

DeToledo et al [40] proposed a numerical and a topological approach for reconstructing implicit information, recovering original data using reverse-engineering algorithms. They achieve high effectiveness, reconstructing 90% of information from massive models with millions of triangles after few minutes of processing.

Yang and Yonghua [41] presented a reverse engineering methodology that is based on haptic volume removing. In their method, a physical object which is to be digitized is buried in a piece of virtual clay that is generated with the help of a fixture. Digitizing

the physical object is performed by simply chipping away the virtual clay with a position tracker that is attached to a haptic device. This method has eliminated the need to merge point clouds that are digitized from different views using current digitizers.

Jun [42] proposed a method that can automatically fill complex polygonal holes on triangular meshes with missing data with a piecewise manner, providing robust results even in regions of high curvature, where other techniques would fail to fill the holes smoothly.

Fayolle et. al. [43] 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. [44] 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.

Sithole and Vosselman [45] 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.

Jeong et al. [46] 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.

Attene and Spagnuolo [47] 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.

Au and Yuen [48] 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.

Ma and Kruth [49] employed a two-step linear approach to fit NURBS curves and surfaces using the measured points of a physical part. The weights of the control points are first identified from a homogeneous system using symmetric eigenvalue decomposition, and the control points are further processed in a way similar to B-Spline curve and surface fitting.

Amenta et al. [15] 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 with many application in computer graphics. The main objective of these methods is to provide a triangular mesh or another smooth surface representation, mainly for rendering purposes. Some of them allow the extraction of characteristics in the context of identifying features on the model [44, 45, 46, 48]. However, they do not provide any means of editing, other than interactively altering the positioning of triangle vertices.

On the other hand, a feature that came to our attention concerned the physical primitives which control the behavior of a model during a deformation caused by a force applied upon the model, as discussed in methods using deformable models [50, 51, 52, 53, 54]. These methods consider the behavior of a model in a motion estimation, or the recognition of features in various states of the model, but do not offer a context of editing given a tool set of editing operators.

Based on these two considerations, we were motivated to extend the line of research by making use of digitally reconstructed models, such as described in the above-mentioned methods, which would also have properties as described in the methods of deformable models, by including the option for editing and re-manufacturing in the context of computer aided design. The contribution of our work is the introduction of an editing toolset, which will allow the CAD end user to perform meaningful modifications on the model.

A discussion on related work is also available in the following sections, where we cite on research concerning the scope of each of the remaining chapters exclusively.

2.2 Slicing and Segmentation

Dolenc and Makela [55] studied several methods of slicing models for layered manufacturing, with the objective to minimize the number of layers while preserving the accuracy of the model. They also handle flat areas and restrict the staircase effect to a user-specified tolerance.

Jamieson and Hacker [56] compared the approach of slicing CAD models with tessellation techniques for use in the rapid prototyping industry.

Kulkarni and Dutta [57] describe two factors associated with the slicing procedures used in layered manufacturing processes that introduce geometric inaccuracy, and suggest solutions for their redressal.

Sabourin et al [58] propose a method for refining the slicing thickness adaptively through interpolation rather than extrapolation, which is well suited for execution in a parallel processing computer.

Hope et al [59] developed an adaptive slicing procedure, which uses layers with sloping boundary surfaces that closely match the shape of the required surface. This greatly reduces the stair case effect which is characteristic of layered components with square edges.

Tata et al [60] proposed an adaptive slicing method, in which the layer thickness is adjusted such that one of the four criteria is met, i.e. cusp height, maximum deviation, chord length and volumetric error per unit length.

Tyberg and Bohn [61] presented an approach for reducing the fabrication times by eliminating most of the unnecessary layers that do not effectively enhance the overall quality of the part surfaces.

Mani et al [62] describe a slicing technique that permits the fastest layer manufacture of an object that has different surface finish requirements on different surfaces. The user can select surfaces of the model and for each impose a distinct cusp height requirement.

Other works that handle the slicing of a model include [63, 64, 65, 66, 67]. Our approach follows the main concepts of [68, 69, 70], where the importance of parameters like the slicing direction or the slice thickness is addressed. These works also discuss the adaptive computation of the ideal slicing thickness. Since this is not in the scope of our work, to avoid exhaustive re-computations, we chose the option of user-specified thickness, which however does not limit the generality of our method.

2.3 Feature Poly-lines

Several methods have been proposed that include the representation of an ordered or unordered point set with smooth continuous curves, both in 2D as well as in 3D.

Renner [3] developed a curve fitting method to represent curves occurring mainly in mechanical engineering practice by connecting sections that retain tangent vector continuity. The interpolation is local in the sense that the shape of the curve at a certain data point is influenced by four other data points in its vicinity.

Dyn et al [71] suggested a simple interpolation scheme based on a 4-point recursive subdivision for curve and surface design, that provides C^1 continuous curves for a certain range of a given tension parameter.

Fang and Gossard [72] presented a method for generating a piecewise continuous parametric curve from a set of unordered and error-filled data points. The resulting curve not only provides a good fit to the original data but also possesses good fairness.

Pigounakis and Kaklis [73] developed a two-stage automatic algorithm for fairing C^2-continuous cubic parametric B-splines under convexity, tolerance and end constraints, which preserves the convexity and end properties of the output of the first stage and, moreover, it embodies a global tolerance constraint.

Lee [74] presented an algorithm to approximate a set of unorganized points with a simple smooth curve without self-intersections, in which a moving least-squares technique is suggested using Euclidean minimum spanning tree, region expansion and refining iteration.

Liang et al [75] developed a free form shape representation technique using Non-Uniform Rational B-Spline (NURBS) modeling, and the accuracy of the representation is evaluated by using a centroid-radii error function, which computes the cumulative distance between the intersection points by radii lines on the boundary of the original image and the reconstructed image.

Said [76] introduced the concept of the alternating convex hull, which is used to develop a new method that generate poly-line approximation for single-valued digital curves. The suggested method has the ability to deal with closed curves if they decomposed to two single-valued curves. The method is hierarchical in nature, that is, the approximation varies from coarser to finer, using nested alternating upper/lower hulls, to comply with error tolerance specified.

Wang et al [77] presented a simple and efficient technique to generate approxi-

mately arc-length parameterized spline curves that closely match spline curves typically used to model roads in high-fidelity driving simulators.

Chen et al [78] presented a method for automatically generating an interpolation closed G^1 arc spline on a given closed point set, in which the point set is treated otherwise for an even number of points than an odd number of points. For odd point sets one of the exactly two different closed interpolating G^1 arc splines is chosen, while for even point sets the explicit interpolation condition for generating closed G^1 arc splines is generated, and a weight function for each given point is defined. The points are automatically chosen and moved based on weight functions such that the interpolation condition is fulfilled, and the G^1 arc splines are constructed such that the radii of the arcs in the spline are close to each other.

Sabin and Dodgson [79] improved the four-point subdivision scheme, which had rather large longitudinal artifacts and points interpolated around a curve of almost constant curvature were fitted by a curve with significant variations of curvature. They describe a geometry-sensitive variant of this scheme which does not have this problem. In fact circles are reproduced exactly with any spacing of the initial data.

Azariadis and Sapidis [80] focused on drawing curves on a cloud of points by constructing a rough design composed of poly-lines which only look like they lie onto the cloud surface. Their system replaces these poly-lines by smooth curves lying exactly onto the cloud surface, while the intermediate side effects like the line wrinkles that appear in the process are eliminated.

Dyn et al [81] constructed another four-point subdivision scheme, which generates C^2 curves and reproduces cubic polynomials. The refinement rule of this scheme is obtained by constructing, for each interval (or edge) in the coarser level, a cubic polynomial that interpolates the four points closest to the interval, and then evaluating this polynomial at $1/4$ and $3/4$ of the interval. The collection of the pairs of points, corresponding to each edge in the coarser level, constitute the refined set of points.

Huang et al [82] developed a simple algorithm for simultaneous degree elevation and knot insertion for B-spline curves, which can handle unclamped B-spline curves. Their method is based on the simple approach of computing derivatives using the control points, re-sampling the knot vector, and then computing the new control points from the derivatives.

Sanchez-Reyes and Fernandez-Jambrina [83] analyzed the chord-length parametrization using bipolar coordinates and compared their technique with the arc-length

parametrization for curve fitting.

Dyn et al [84] replaced the uniform parameter values of a four-point subdivision scheme by chordal and centripetal parameter values, to provide two new non-linear schemes that are convergent and bound the distance between the smooth limit curve and the initial control polygon.

Li et al [85] proposed the arterial snake, a deformable primitive, to decompose a detail surface into a multi-layer collection of 1D generative curves. They offer a purely geometric approach to extract the arterial patterns and define their layering relationship leading to an enhanced reconstruction of the input scans, and a simple, intuitive, yet powerful shape manipulation framework that allows easy and natural editing and creation of plausible and realistic variants of the original geometry.

Our work has been influenced by the methods described in [76, 3, 86, 87, 88], so there are several elements that have been implemented according to their suggestions as a whole, or as variations that serve our own intends. For example, [76] discusses an iterative method that uses the convex hull in all iterations, but we only use it in our first iteration, and proceed with the Voronoi diagram for the remaining iterations. The reason for switching methods is discussed in section A.3 of the Appendix.

2.4 Surface Reconstruction

Several methods have been developed that reconstruct the surface of a model 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.

Kels and Dyn [89] developed a multi-resolution method based on iterative refinement of the sets representing the cross-sections, with a geometric weighted average of two sets, defined for positive weights (corresponding to interpolation) and when one weight is negative (corresponding to extrapolation). To obtain a smoother reconstruction of the 3D object, they adapt to sets the 4-point interpolatory subdivision scheme.

Pal [90] used geometric subdivision and NURBS interpolation to achieve accurate

shape building using scanned data, manufacturing ability of complex shapes, faster and accurate shape representation with high quality surfaces, model portability, and a better control on object shape and better patch-planning.

Liu et al [91] consider the more general problem where input data may lie on non-parallel cross-sections and consist of curve networks that represent the segmentation of the underlying object by different material or tissue types (e.g., skin, muscle, bone, etc.) on each cross-section. The desired output is a surface network that models both the exterior surface and the internal partitioning of the object. They introduced an algorithm that is capable of handling curve networks of arbitrary shape and topology on cross-section planes with arbitrary orientations.

Sangveraphunsiri and Sritrakulchai [92] proposed a two-level adaptive hierarchical clustering algorithm to manage unorganized points, so that the triangular mesh models can be correctly obtained by applying a triangular mesh creation algorithm.

Pal and Ballav [93] developed a method for fitting NURBS surfaces on point cloud data with extreme through-point accuracy, minimum shape and geometry loss, and quicker reconstruction. They prepare a logical rarefied data set out of a dense data set in which data points are sequenced for reconstruction. The logical selection sets are constructed in such a way that these strictly entertain only negligibly small shape losses and ensure a fastest reconstruction process with an optimum number of data points.

Boissonnat and Memary [94] consider the problem of reconstructing a shape from unorganized cross-sections from medical imaging such as free hand ultrasound apparatus. The position and orientation of the cutting planes may be freely chosen, and the input data consist of the cutting planes and their intersection with the object. They compute the arrangement of the cutting planes, and then, in each cell of the arrangement, they reconstruct an approximation of the object from its intersection with the boundary of the cell. Lastly, they glue the various pieces together.

Yang [95] used two nonlinear subdivision schemes, face based subdivision scheme and normal based subdivision scheme, for surface interpolation of triangular meshes. With a given coarse mesh more and more details are added to the surface when the triangles have been split and refined. Because every intermediate mesh is a piece-wise linear approximation to the final surface, the first type of subdivision scheme computes each new vertex as the solution to a least square fitting problem of selected old vertices and their neighboring triangles. Consequently, sharp features as well as

smooth regions are generated automatically. For the second type of subdivision, the displacement for every new vertex is computed as a combination of normals at old vertices. By computing the vertex normals adaptively, the limit surface is G^1 smooth. The fairness of the interpolating surface can be improved further by using the neighboring faces. Because the new vertices by either of these two schemes depend on the local geometry, but not the vertex valences, the interpolating surface inherits the shape of the initial control mesh more fairly and naturally.

Ohtake et al [96] proposed a method for approximating an unorganized set of points scattered over a piecewise smooth surface by a triangle mesh, by generating an adaptive spherical cover and auxiliary points corresponding to the cover elements. The intersections between the spheres of the cover are analyzed and the auxiliary points are connected, while the resulting mesh is cleaned from non-manifold parts. The method allows to control the approximation accuracy, process noisy data, and reconstruct sharp edges and corners.

Lee [97] worked with generalized cylinders defined by contours of discrete curves, and proposed two algorithms to generate generalized cylinders surfaces in polygonal meshes and in developable surface patches of the cylindrical type. To solve the contour blending problem of generalized cylinder, he adopted the algorithms and properties of linear interpolation by direction map that interpolate geometric shapes based on direction map merging and group scaling operations.

Kuo and Yau [98] presented a Delaunay-based region-growing surface reconstruction algorithm that holds the advantages of both Delaunay-based and region-growing approaches. The proposed algorithm takes a set of unorganized sample points from the boundary surface of a three-dimensional object and produces an orientable manifold triangulated model with a correct geometry and topology that is faithful to the original object. Their algorithm requires only one-pass Delaunay computation and needs no Voronoi information because it improves the non-trivial triangle extraction by using a region-growing technique. It also makes the surface reconstruction more systematic and robust because it inherits the structural characteristics of the Delaunay triangulation, which nicely complements the absence of geometric information in a set of unorganized points.

Karniel et al [99] presented a practical solution for surface fitting problems with prioritized geometry constraints in reverse engineering. Their approach allows prioritizing constraints and uses them for decomposing the problem into a set of sequen-

tially solved, manageable sub-problems. The result of each solution step is trade-off between satisfying the set of constraints and fitting of the surfaces to the measured points. The overall solution process trades off solution quality with complexity of the problem.

Tam et al [100] developed a method for computing offsets of profiles on three-dimensional surfaces based on processing the intersections of the offset segments. Their method does not assume a particular curve representation of the profiles, and it covers both closed and open profiles. A two stage approach of first constructing the local and then the global offset of the progenitor profile is adopted. Conditions governing the construction of those offsets are used for the construction.

Svitak and Skala [101] studied the problem of surface reconstruction from sets of planar parallel slices representing cross sections through 3D objects, based on the correct estimation of the structure of the original object. They focus on the structure determination of the 3D object, and their approach is based on considering mutually orthogonal sets of slices.

Peternell and Steiner [102] used as main geometric features of their modeling system the detection of planar faces for the generation of a CAD model of buildings from airborne laser scanner data.

Lin et al [103] presented an algorithm for reconstructing a triangle mesh surface from a given point cloud. Starting with a seed triangle, the algorithm grows a partially reconstructed triangle mesh by selecting a new point based on an intrinsic property of the point cloud, namely, the sampling uniformity degree. The reconstructed mesh is essentially an approximate minimum-weight triangulation to the point cloud constrained to be on a two-dimensional manifold.

Kolluri et al [104] introduced a noise-resistant algorithm for reconstructing a watertight surface from point cloud data, which forms a Delaunay tetrahedralization, then uses a variant of spectral graph partitioning to decide whether each tetrahedron is inside or outside the original object. The reconstructed surface triangulation, which is the set of triangular faces where inside and outside tetrahedra meet, can produce manifold surfaces.

Geng et al [10] 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.

Fayolle et al [43] used an approach, where standard shapes and relations are interpreted as primitives and operations of a constructive model. The input information provided by the user is a sketch model, where the construction tree contains only specified operations and types of primitives while the parameter values of operations and primitives are not defined and recovered by fitting.

Azariadis [105] used dynamic base surfaces that are dynamically adapted to the three-dimensional shape implied by the clouds of points. The only assumption regarding the cloud of points is the existence of a boundary defined by a closed path of four curves. The proposed method is based on an iterative procedure where a dynamic base surface is gradually improved approximating more faithfully the fundamental geometry of the cloud of points. Parameterization is achieved by orthogonally projecting the cloud of points onto the dynamic base surface.

Ivrissimtzis et al [106] studied the use of neural network algorithms in surface reconstruction from an unorganized point cloud, and meshing of an implicit surface. Their algorithm works by sampling randomly a target space, usually a point cloud or an implicit surface, and adjusting accordingly the connectivity of the neural network. This solution gives satisfactory results in sharp features and concavities, and its speed is virtually independent from the size of the input data, making it particularly suitable for the reconstruction of a surface from a very large point set.

Ohtake et. al. [44] offered a shape representation called the multi-level partition of unity implicit surface, which uses piecewise quadratic functions that capture the local shape of the surface, weighting functions that blend together these local shape functions, and an octree subdivision method that adapts to variations in the complexity of the local shape.

Dey and Goswami [107] described a simple algorithm called Tight Cocone, which works on an initial mesh generated by a popular surface reconstruction algorithm and fills up all holes to output a water-tight surface. Their method does not introduce any extra points and produces a triangulated surface interpolating the input sample points.

Piegl and Tiller [108] studied and analyzed several problems on skinning techniques and proposed a solution that avoids all the anomalies at the expense of increasing the number of control points and the compute time.

Jeong et al. [46] 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 [48] 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. [15] 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.

Weng et al [13] 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.

A survey on methods for reconstructing surfaces from unorganized point sets is also available in [109] where several known methods are evaluated.

These approaches are very interesting and have found several applications in computer graphics. When it comes to reconstructing the surface of a 3D model from its cross sections, many approaches use contour triangulation methods to obtain the 3D model:

Kels et al [110] applied tools of computational geometry, segment Voronoi diagrams and planar arrangements to the computation of the metric average of 2D sets with piecewise linear boundaries. Such sets are collections of simple polygons and of simple polygons with holes. Since a compact 2D set with boundary consisting of closed curves can be linearly approximated by a 2D set with piecewise linear boundaries, their algorithm provides a computational method for approximation of set-valued functions with 2D images from a finite number of samples.

Whited and Rossignac [111] defined the b-compatibility for planar curves and proposed new ball morphing techniques (called B-morphs) between pairs of b-compatible curves. B-morphs use the automatic ball-map correspondence from which they derive

vertex trajectories (Linear, Circular, Parabolic). They also provided simple constructions for these b-morphs using the maximal disks in the finite region bounded by the two curves.

Peiro et al [12] 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.

Braude et al [112] presented a technique for creating a smooth, closed surface from a set of 2D contours, which interprets the pixels that make up the contours as points in R^3 and employs Multi-level Partition of Unity (MPU) implicit models to create a surface that approximately fits to the 3D points. MPU implicit models provide a superior approach to the problem of contour-based surface reconstruction, especially in the presence of noise, because they are based on adaptive implicit functions that locally approximate the points within a controllable error bound.

Barequet and Vaxman [113] offered a method to incorporate the influence of more than two slices at each point in the reconstructed surface. They investigated the flow of the surface from one slice to the next by matching vertices and extracting differential geometric quantities from that matching. Interpolating these quantities with surface patches then allows a nonlinear reconstruction which produces a free-form, non-intersecting surface.

Ju et al [114] presented a method that automatically constructs a 3D surface network from 2D curve networks with arbitrary topology and partitions an arbitrary number of materials. The surface network exactly interpolates the curve network on each plane and is guaranteed to be free of gaps or self-intersections. Their method also provides a flexible framework for user interaction so that the surface topology can be modified conveniently when necessary.

Huang et al [115] developed a fast triangulation algorithm from planar contours (FTA), in which the judgment of the similarity of contours is carried out, followed by a traditional global optimization method that is applied to triangulating dissimilar contours. A local optimization method is also applied on areas enveloped by line sections of similar contours.

Barequet et al [116] presented a method for interpolating a piecewise-linear surface between two parallel slices, each consisting of an arbitrary number of polygons that define 'material' and 'non-material' regions. Their method is based on computing cells in the overlay of the slices that form the symmetric difference between them. Then, the straight skeletons of the selected cells guide the triangulation of each face of the skeletons. The resulting triangles are lifted up in space to form an interpolating surface.

Akkouche and Galin [117] addressed the problem of reconstructing of a three-dimensional object from cross-sectional contours. Their technique based on a stratification of polygons and anisotropic distance functions that fully exploit the partial structure of the data. A potential field function is created for each cross-section and they are combined to create an implicit surface that contains the contours.

Cong and Parvin [118] proposed an approach for surface recovery from planar sectional contours, based on the so called 'equal importance criterion', which suggests that every point in the region contributes equally to the reconstruction process. The problem is formulated in terms of a partial differential equation, and the solution is calculated from distance transformation. The proposed technique allows for surface recovery at any desired resolution, thus avoiding the inherent problems of correspondence, tiling, and branching.

Barequet et al [119] use the slopes of the previously computed triangles created in the interpolation of neighboring layers to guide the interpolation of the current layer, to achieve smoothly connected consecutive layers.

Earlier works on contour triangulations have been published in [120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138], with the methods of Keppel [138] and Fuchs [137] being the most designating works.

2.5 Editing

Several methods have been developed that deal with editing 3D objects in various ways. Depending on the issue they are addressed to, each method uses another approach to deal with the corresponding problem.

Andrews et al. [139] have developed a user-guided method to capture the shape of a boundary mesh representation as a well-structured, parameterized, procedural

geometry description that can be fine-tuned to meet the current specifications and design requirements. Their framework consists of an integrated system of geometrical extraction and fitting routines, which are applied to user-designated portions of a given input, combined with a set of traditional 'forward' 3D modeling tools that allow the extracted geometry to be edited and fine-tuned.

Chen et al. [140] presented a fast and precise method for extracting bas-reliefs based on differential coordinates, and also provide an editing tool specifically designed for relief editing. They estimate the base surface using normal smoothing and Poisson reconstruction, and extract the relief by height thresholding. The editing tools they provide allow for global transformations (translation, rotation, and scaling) of the whole relief, as well as local modifications to the relief.

Gingold and Zorin [141] describe a sketch based modeling technique based on changing shaded images of three dimensional models directly, using free form strokes for two dimensional image editing. The shape is automatically adjusted to match desired changes of appearance by minimizing a quadratic functional with tangent and positional constraints deduced from user image modifications.

Wand et al. [142] thought of a multi-resolution data structure for real-time visualization and interactive editing of large point clouds. They provide interactive editing tools for large scanner data sets. Their data structure provides efficient rendering and allows for handling very large data sets using out-of-core storage. It also provides dynamic operations for on-line insertion, deletion and modification of points with time mostly independent of scene complexity. This permits local editing of huge models in real time while maintaining a full multi-resolution representation for visualization.

Zimmerman et al. [143] introduced an over-sketching interface for feature-preserving surface mesh editing, where the user sketches a stroke that is the suggested position of part of a silhouette of the displayed surface. The system then segments all image-space silhouettes of the projected surface, identifies among all silhouette segments the best matching part, derives vertices in the surface mesh, selects a sub-region of the mesh to be modified, and feeds appropriately modified vertex positions together with the sub-mesh into a mesh deformation tool.

Kara et al. [144] have developed a system which takes as input a 2D concept sketch of an object, and a generic 3D wireframe template. The template is aligned with the input sketch and the user traces feature edges of the sketch on the computer screen. User's 2D strokes are processed and interpreted in 3D to modify the sketch,

which is refined using physically-based deformation techniques.

Yoon and Kim [145] proposed a sweep-based approach to the free-form deformation of three-dimensional objects. Instead of using a volume enclosing the whole object, they approximate only its deformable parts using sweep surfaces. The vertices on the object boundary are bound to the sweep surfaces and follow their deformation. Several sweep surfaces can be organized into a hierarchy so that they interact with each other in a controlled manner.

Botsch and Kobbelt [146] presented a free-form modeling framework for unstructured triangle meshes based on constraint shape optimization. Their method includes setting various boundary constraints to define a basis function, and moving a 9-dof manipulator object to control the modification. Their technique can handle arbitrary support regions and piecewise boundary conditions with smoothness ranging continuously from C^0 to C^2.

Sorkine et al. [147] have described a set of editing operations, such as interactive free-form deformations in regions of interest based on the transformation of a handle, transfer and mixing of geometric details between two surfaces, and transplanting of a partial surface mesh onto another surface. The main computations involve solving sparse linear systems, which can be done at interactive rates.

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 the user intends to manipulate their model using high level editing methods. The main concept of the method described in this chapter is that it allows editing and re-manufacturing of the model in the context of computer aided design.

2.6 Centerlines

A variety of methods have been proposed that deal with editing 3D objects using several approaches. While editing free-form objects can be done with arbitrary operations, when editing for special cases may require some specific restrictions to be satisfied, such as in the case of medical applications, where the simulation needs to

comply with the behavior of real tissue when a surgery is actually performed.

Chi and Zhang [21] presented an approach for optimizing the raw CT data acquired by a scanner. Their approach transforms the raw area samples to accurate point samples to improve the CT data precision by (a) establishing the mapping relationship between area samples and point samples, (b) segmenting the raw CT slice into different regions based on the human tissue feature, and (c) in each segmented region, constructing quadric spline fitting equations with the mapping relationship, to transform the area samples to more accurate point samples.

Mondy et al. [22] deal with large amounts of image data extracted by Micro-CT datasets using supercomputer imaging technology. Their method investigates the architecture and shared memory requirement of supercomputers needed to generate large models using various intensity levels for accurate data visualization, which is prohibited by desktop computers.

Albu et al. [20] propose a morphology-based approach for inter-slice interpolation of CT and MRI datasets composed of parallel slices. Their method handles explicitly inter-slice topology changes by decomposing a many-to-many correspondence into three fundamental cases: one-to-one, one-to-many, and zero-to-one correspondences. The proposed iterative interpolation process computes a transition sequence between a pair of corresponding input slices, and selects the element located at equal distance from the input slices. The main contribution of their approach is the ability to interpolate between anatomic shapes by creating a smooth, gradual change of shape, and without generating over-smoothed interpolated shapes.

Pekkan et al. [148] developed a surgical planning tool to study the anatomical complexity and patient-specific computational fluid dynamics (CFD) of vascular anatomies. Their method includes two-hand free-form manipulation of a model. The idea is to let the user grab a portion of the shape and then pull, push, twist and bend it. Only the portion of the shape in the vicinity of the grabbed point is affected, and the effect is lessened with distance through a specific decay profile.

Peiro et al. [12] presented a set of procedures for shape reconstruction and triangulation of geometries derived from a set of medical images representing planar cross sections of an object. Their method is based on the interpolation of an implicit function through a set of points obtained from the segmentation of an image set. They also use mesh enhancement techniques to maximize the quality of the triangulation together with curvature adaption to optimize mesh resolution.

Ju et al. [19] presented a method for producing a smooth 3D volume from distorted 2D sections in the absence of undistorted references. Their method is based on pairwise elastic image warps between successive tissue sections, where an average warp is computed for each section from the pairwise warps in a group of its neighboring sections. The average warps deform each section to match its neighboring sections, creating a smooth volume where corresponding features on successive sections lie close to each other.

Sturgeon [17] described a method for building CAD models of bones utilizing medical imaging data. Volumetric data sets obtained from a CT scan provided cross-sectional closed curves, which were combined to form surface patches that described detailed anatomical CAD models with the use of commercial applications.

Sun et al. [18] presented advances of bio-CAD modeling and application in computer-aided tissue engineering, including biomimetic design, analysis, simulation and free-form fabrication of tissue engineered substitutes, and described a methodology for generating bio-CAD models from high resolution non-invasive imaging, the medical imaging process and the 3D reconstruction technique.

Bors et al. [16] proposed an interpolation algorithm for reconstructing an n-dimensional object from a group of $(n-1)-dimensional$ sets representing sections of that object using a mathematical morphology morphing approach. The morphing transformation modifies pairs of consecutive sets such that they approach in shape and size. The interpolated set is achieved when the two consecutive sets are made idempotent by the morphing transformation. The entire object is modeled by successively interpolating a certain number of intermediary sets between each two consecutive given sets.

Weng et al. [13] proposed a surface rendering method using optical flow, the apparent motion in the image plane produced by the projection of real 3D motion onto the 2D images of a medical CT scan. The 3D motion of an object is recovered from the optical-flow field using additional constraints, and is used to extract an accurate surface representation of the 3D model.

2.7 Comparison with our approach

The above-mentioned methods mainly address the issue of reconstructing the surface of the 3D object, but do not provide any tools for editing the model. The editing methods mentioned in section 2.5 provide tools for editing 3D models in various ways, but require as input a model that has previously been structured in a specific format, either in triangle meshes with specified topology [146, 140], or in higher level representations, such as curves or surfaces [144, 145, 143, 141, 139]. Our method provides tools for editing the input model and also can handle unstructured data-sets (i.e. point clouds). We also have the means of extracting the required topology information with the use of the tool-set we describe later on.

In this thesis, we present a method for dividing the point cloud into a set of cross-sections, which we process separately to extract local structural information. This information is used to detect local features of the object. Such features may 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 [149]. This is accomplished by slicing the point cloud into cross sections, and treating each cross section as an individual point set in 2D. Processing each cross section as a 2D set of points allows us to develop very efficient and accurate local feature extraction techniques. These cross sections contain the same information as the initial point cloud - the (x, y, z) coordinates of the points - with the additional information that the points of a slice are located near a planar surface that intersects the object at a specific direction. In other words, the points of each slice can be considered to be co-planar, so we can process each slice as a 2D set of points instead of a 3D object. This provides the means for more efficient and accurate local feature extraction.

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 poly-line, as it is a closed curve of continuity G^0. Afterwards, a closed cubic B-Spline curve is constructed, with continuity G^1, which interpolates the poly-line and provides a smooth boundary representation.

The local per slice feature representation is then combined with information pro-

vided from several adjacent slices, to reconstruct the global structure and morphology of the object. 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 framework for creating an editable model based on B-spline contours without the use of parametric surface patches that is capable of supporting arbitrary tessellations.

Chapter 3

Segmentation of the Point Cloud

3.1 About Point Clouds

3.2 Segmentation of the Point Cloud into Cross Sectional Slices

3.2.1 The Slicing Direction

3.2.2 Thickness of the Cross Sections

3.2.3 Empty Space between Cross Sections

3.2.4 Other Cross Sectional Options

3.3 Treating Cross Sections as 2D Point Sets

3.4 Examples

3.1 About Point Clouds

The first step of our method is to import a data-set, which will be used to build a model of a 3D object. Then we proceed with the segmentation of the data-set, so as to provide a series of point sets that may be processed individually and/or in parallel.

The input format for our method is relatively simple. No assumptions are made about the geometry or the topology of the object. The simple unstructured form of data-set we use is the point cloud of its surface scan. The only information a point cloud carries is the $[x, y, z]$ coordinates of a set of points that lie on the surface of the model, as illustrated in figure 3.1. Depending on the acquisition method used and the density of the scan, a point cloud may describe the topology of the boundary accurately at all parts of the model (e.g. 3D laser scanners) [9, 8], or describe only

feature points of the object that lie on specific parts of the model (e.g. medical CT scans) [150, 21]. The case of medical CT scans is considered delicate, as the direction and the density of of the scan are predefined, and cannot be refined after the data-set has been acquired. To begin processing the input we have acquired, we proceed to the segmentation of the point cloud into several cross sectional slices. In this chapter we discuss the details of this segmentation process.

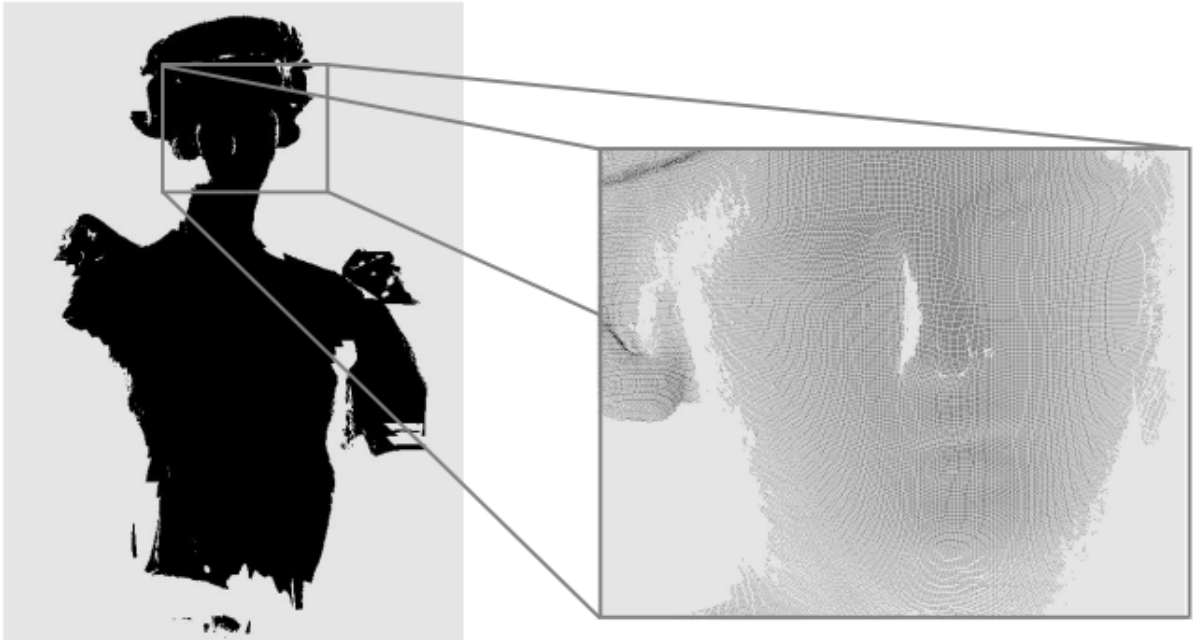


Figure 3.1: The point cloud of a female figure. The only information provided are the $[x, y, z]$ coordinates of points on the surface of the model.

For simplicity of the computations, all point clouds used in this study have a common characteristic, i.e. they are all some type of generalized cylinders. In other words, our models do not have parts that form branches, and they also there are no holes present (all objects are of genus 0). For this type of objects, the complexity of constructing a skeleton for our model is minimal. However, the user requirements usually demand the processing of complex objects, which may have holes in their surface ($genus > 0$), or have a complex geometry with branches and parts that cannot be described with generalized cylinders. In such cases, the method we propose will not work as it is. But we do have the option to split such objects into smaller parts (that form generalized cylinders) and treat each part as a discrete object. For example, we could adopt a method such as the one proposed by [151], which is also discussed in [152] that uses a Reeb Graph to determine the regions where a model could be

segmented, to acquire perceptually meaningful components that suit our needs. This, however, may be done as a pre-processing step before we apply our method, and therefore is not discussed in this study.

3.2 Segmentation of the Point Cloud into Cross Sectional Slices

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. The idea is to traverse the point cloud from one side to the other, using a moving planar surface that intersects the point cloud. Each of these cross sections is defined as a thin slice of the point cloud, which consists of several points of the cloud that are located near the current plane, as illustrated in figure 3.2. 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.

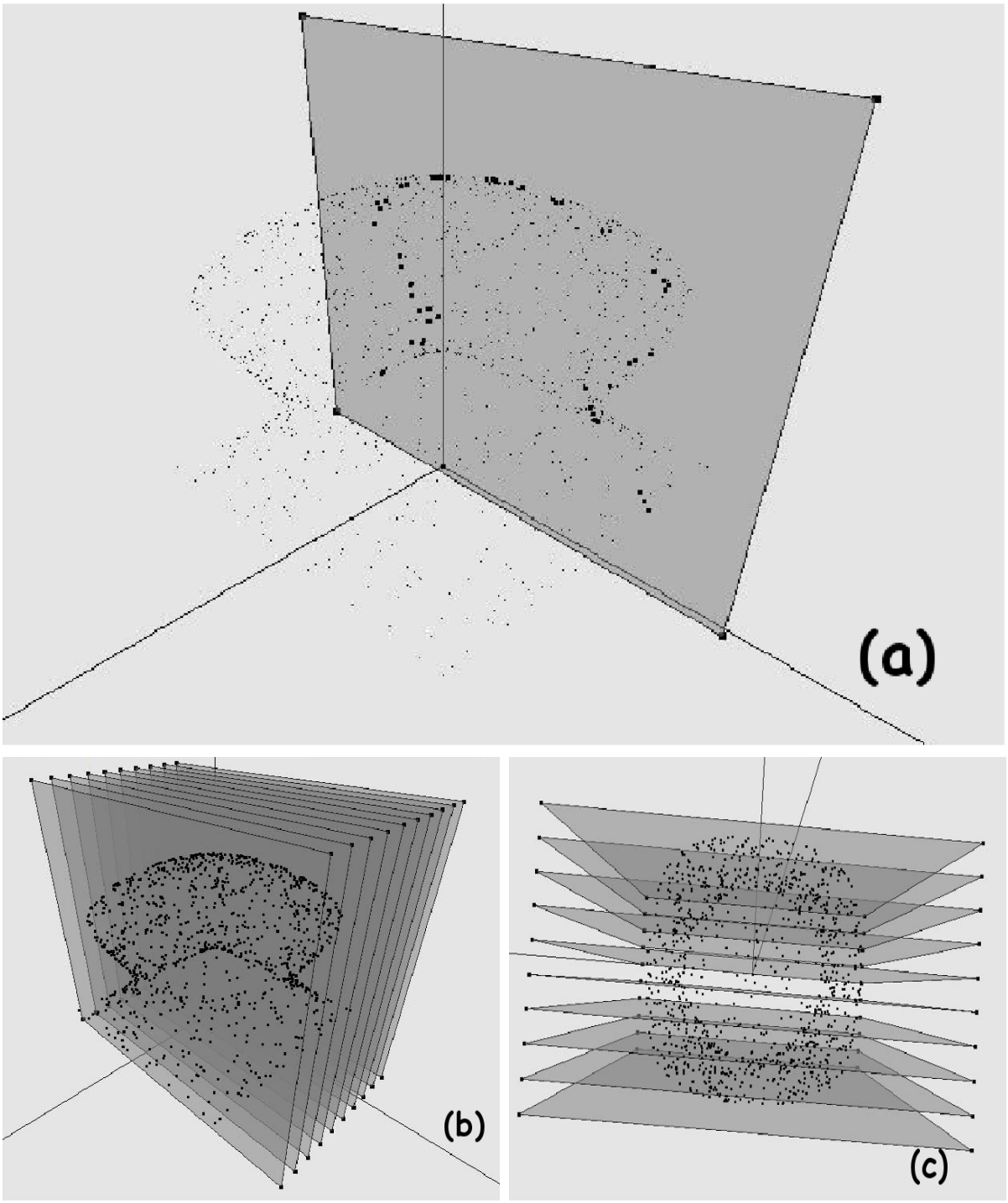


Figure 3.2: (a) The point cloud of a salt dome with one cross section highlighted. (b) The point cloud is divided into several cross sections. (c) Top view of the cross sections.

To slice the point cloud into cross sections, we need to consider a number of parameters that affect the overall computations as well as the quality of the resulting model. Such parameters are the direction of the cross sections, the thickness of each

cross section, and the distance between cross sections. They can be either specified by the user, or calculated automatically.

3.2.1 The Slicing Direction

The first parameter we discuss for the point cloud segmentation is the slicing direction. The direction along which we choose to divide the object in slices may influence the process of feature extraction. To illustrate this, consider an object that contains a cylindrical feature, e.g. a hole in the shape of a cylinder, such as the one in figure 3.3. If the slicing direction matches the cylinder axis, the points of the slice located near the hole would form a circular region (figure 3.3a). If the slicing direction is different than the cylinder axis, the points of the slice would form an elliptical region (figure 3.3b), a rectangle (figure 3.3c) or even a single line segment (figure 3.3d).

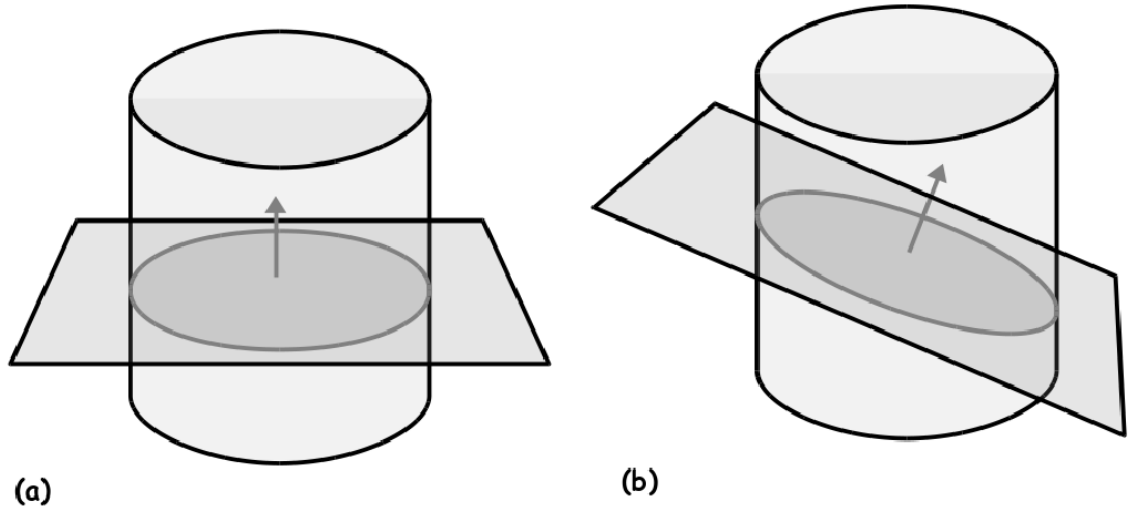


Figure 3.3: The slicing direction is an essential parameter for the modeling process. (a) The slicing direction matches the cylinder axis. The points of a cross section form a circle. (b) The slicing direction is different than the cylinder axis. The points of a cross section form an ellipse.

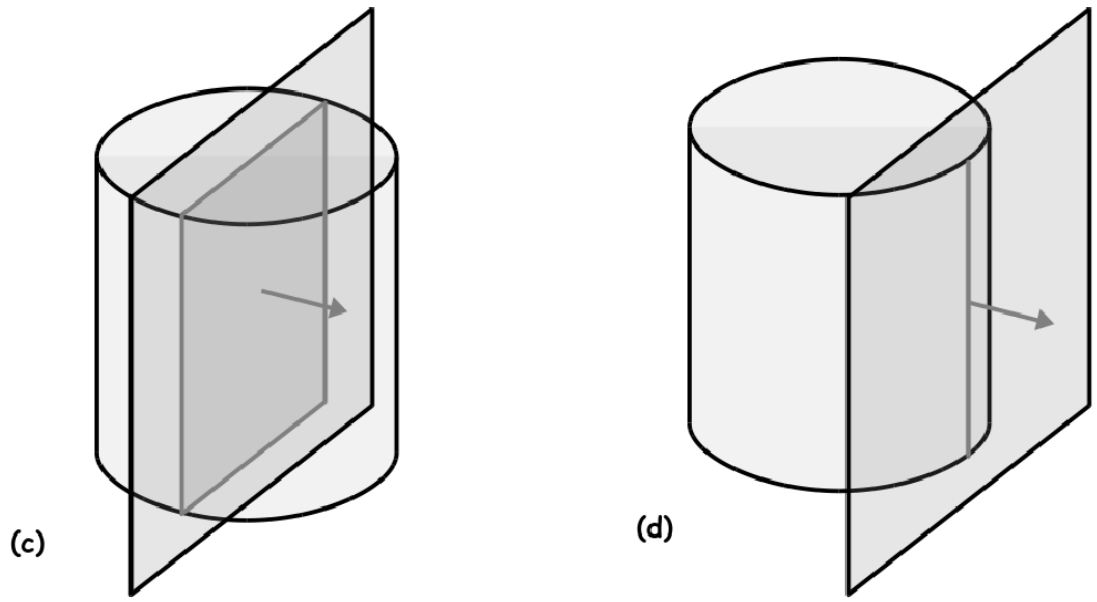


Figure 3.3: The slicing direction is an essential parameter for the modeling process. (c) The slicing direction is vertical to the cylinder axis. The points of a cross section form a rectangle. (d) The slicing direction is vertical to the cylinder axis. The points of the last cross section form a line.

The proper slicing direction is related to the medial axis of the object. For simple objects which have no branches, joints or protrusions of any type, the medial axis consists of a simple curve. A direction following the average path of this curve usually satisfies the user requirements. For complex objects that have branches, the medial axis consists of a set of curves that form a kind skeleton of the object. Such point clouds can be divided into simpler parts, which are treated as independent point clouds, and each part is sliced according to its proper direction. In our tests we have only experimented with point clouds that have no branches, whereas treating objects with branches is not at the scope of this thesis. To align the point cloud to the proper direction, we perform principal component analysis [153] using the ALGLIB numerical analysis library [154]. For simplicity of the computations, the entire point cloud is then realigned so that the principal axis matches the z-axis of the environment.

3.2.2 Thickness of the Cross Sections

The thickness of each cross section also affects the quality of the resulting model. The segmentation of the point cloud into cross sectional slices inevitably results in loss

or distortion of information locally. However, if a proper slicing thickness is carefully chosen, the information corruption is negligible. In other words, having very thick slices will result in cross sections where the projected 2D point set differs significantly from the initial 3D point set. Moreover, the points of a very thick slice may not provide useful information, as we might get many features tangled together (see figure 3.4b). To avoid this issue we should use thinner slices (figure 3.4a). On the other hand, a set of very thin slices may contain too few points and may be inadequate to describe a feature (figure 3.4d), or there may be adjacent slices that carry almost the same information.

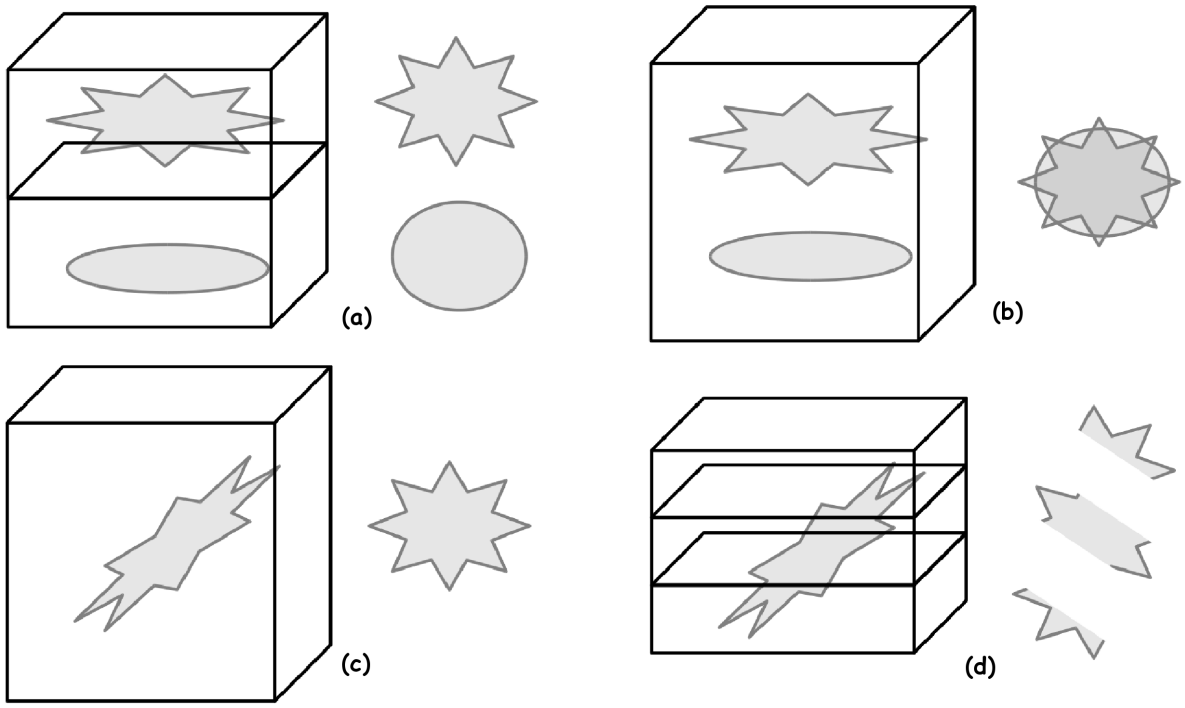


Figure 3.4: The thickness of the slices is another essential parameter. (a) Each slice is required to provide the proper information about the boundary of the model in its context. (b) Very thick slices may result in having features tangled together. (c) A proper slice thickness may eliminate problems caused by an inappropriate slicing direction. (d) Very thin slices may fail to describe a feature due to lack of information.

Since we cannot be certain if the thickness we have chosen is satisfactory for the following computations, we have included an option in our demo implementation that allows the user to split a cross section into two new cross sections with reduced thickness, or to merge two or more cross sections into one cross section with increased thickness.

Objects in which the surface has an increased level of detail, such as rough terrains, should be sliced into very slim cross sections, so as not to loose the details when projecting the points of the cross section. For objects with a smooth surface the cross sections should be thicker, because if we take thin cross sections, we may end up with several adjacent cross sections that provide the same information. Of course, the thickness of the cross sections may vary from slice to slice, as some parts of the point cloud may have detailed surface and require thinner cross sections than other parts with smoother surface.

As described in [70, 63, 155, 65], the ideal slice thickness may be determined adaptively. However, to avoid performing a morphological analysis of the 3D point cloud in the preprocessing phase, the user is also allowed to manually set the thickness of the cross sections and segment the point cloud into slices of equal thickness.

3.2.3 Empty Space between Cross Sections

Another option for the user is to have some empty space between two adjacent cross sections. There may be cases where some empty space is required between some cross sections, e.g. to intentionally omit parts of the point cloud from being processed. For example, a part of the object may contain a local feature we do not intend to include in our model. We have the option to leave this feature out of our computations by considering this part as empty space. 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.

The projection of the cross section on its corresponding 2D plane produces some space between the cross sections that is empty. The information of this part, however, is not omitted from the computations, as it has also been projected on the 2D plane, so this space does not count as empty space between the cross sections. In our examples, we set each slice to start exactly where the previous slice ends, as we do not want to omit any information from points located between two slices (i.e. there is no empty space between adjacent slices).

3.2.4 Other Cross Sectional Options

As we mentioned earlier, the user is allowed to slice the point cloud into cross sections of variable thickness, so we may have thick slices on one part of the model and thin slices on another. This means that the thickness is a parameter built in the data structure of each cross section.

There is also the option to merge two or more cross sections into one cross section with increased thickness, or to split a cross section into two cross sections with half thickness. To perform such operations, one or more cross sections have to be selected. For this feature, an option is included in the data structure of the cross section, which allows the cross section to be selected. One or more cross sections can be selected at once, even if they are not adjacent. This feature will be useful later on, when we perform modifications on our model. Another option is the ability to define a new cross section between two existing cross sections. This feature is discussed in detail in section 5.2.

3.3 Treating Cross Sections as 2D Point Sets

Once we have segmented the point cloud into cross sections, the points that belong to each cross section are projected on a plane that is vertical to the slicing direction. This will allow us to use 2D techniques for processing the points of the slice. Since we have aligned the principal axis of the point cloud to match the z-axis of the environment, the projection is easily achieved by setting the z coordinate of all points equal to z_0 of the corresponding plane. If this is not the case, i.e. the principal axis does not match the z-axis of the environment, the projection is calculated according the position and the direction of the 2D plane.

A known methodology for achieving this is to align the points of a cross section e.g. to the $z - axis$, set $z_i = z_0$ for all points and then re-align the points to their previous position:

i. Translate points so that the projection plane passes through the origin (any point of reference on the target plane, e.g. the point with $x = 0, y = 0$).

ii. Rotate points about the z axis so that the rotation axis lies in the xz plane.

iii. Rotate points about the y axis so that the rotation axis lies along the z axis.

iv. Set $z = 0$ for all points of the cross section.

v. Apply the inverse of step iii.

vi. Apply the inverse of step ii.

vii. Apply the inverse of step i.

At the end of this process all points of a cross section will be projected to the their corresponding $2D$ plane.

An example of projecting the cross sections is shown in figure 3.7, in which the points on part (b) of the cross section are projected to the corresponding plane, while the points on part (a) of the cross section remain unprojected. The projection of the points to their 2D plane makes the point set of the cross section co-planar. This allows us to perform operations in the 2D space instead of 3D, making the computations simpler.

3.4 Examples

Figures 3.5, 3.6 and 3.7 illustrate some examples of a sliced point cloud of a Screwdriver with 63690 points from Cyberware [1]. In this example, the slicing direction is obvious, as the principal axis corresponds to the direction of the screwdriver shaft.

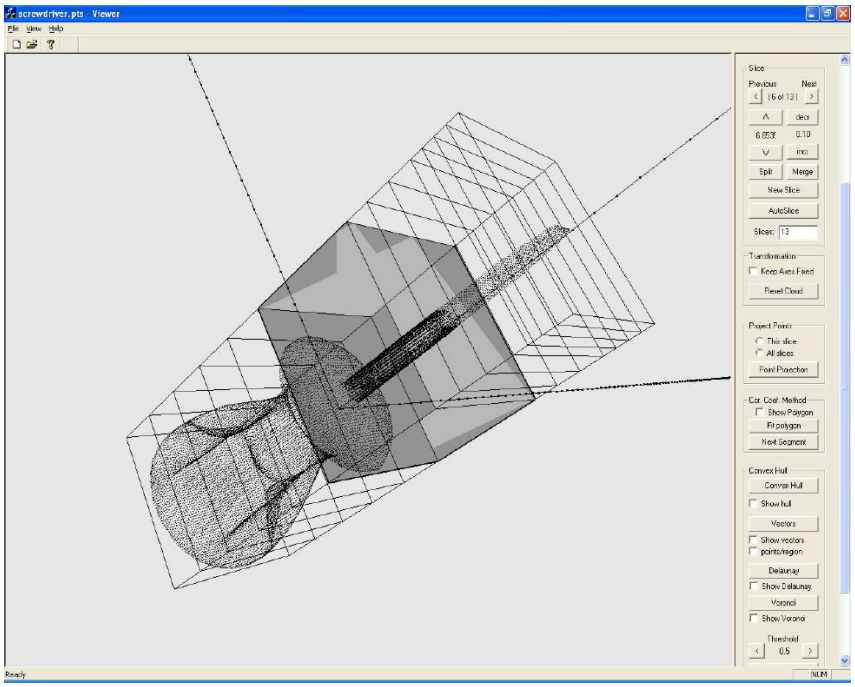


Figure 3.5: The Cyberware Screwdriver. The point cloud may be sliced into cross sections of variable thickness. The principal axis of the point cloud usually matches the ideal direction for slicing the cloud.

Although the slicing direction seems obvious in this example, it may be difficult to determine the proper thickness of each slice. This has to do with the level of detail on the surface of the model. Some parts of the model may have simple features with no details, such as the shaft of the screwdriver, which forms a cylinder with no details. Other parts may have features with more details, such as the handle of the screwdriver, which has a more complex shape. On the area of the shaft it is considered safe to take cross sections of increased thickness. The area of the handle requires thinner cross sections, to avoid having many points on each slice, which implies increased loss of information. Figure 3.6 shows various instances of the slices screwdriver and a single slice on the handle with different amounts of thickness. The quality of the model decreases and the loss of information is greater as we increase the thickness of the cross sections.

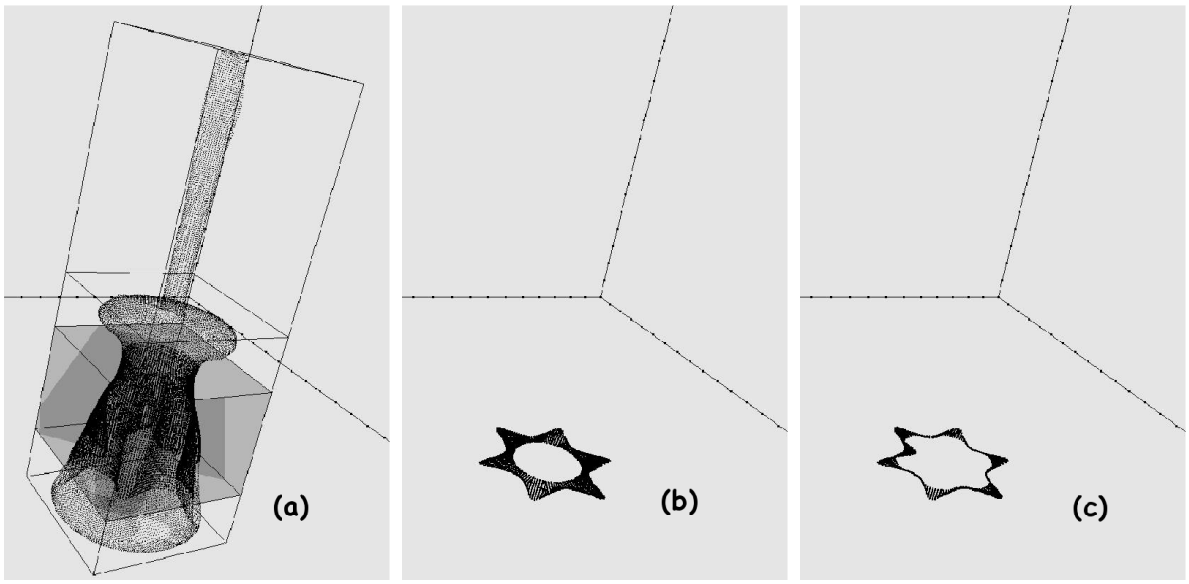


Figure 3.6: Instances of the sliced screwdriver with different values of thickness. When the highlighted cross section from (a) is projected on its 2D plane, we get the slice in (b). Having cross sections with half its thickness will project to slice (c).

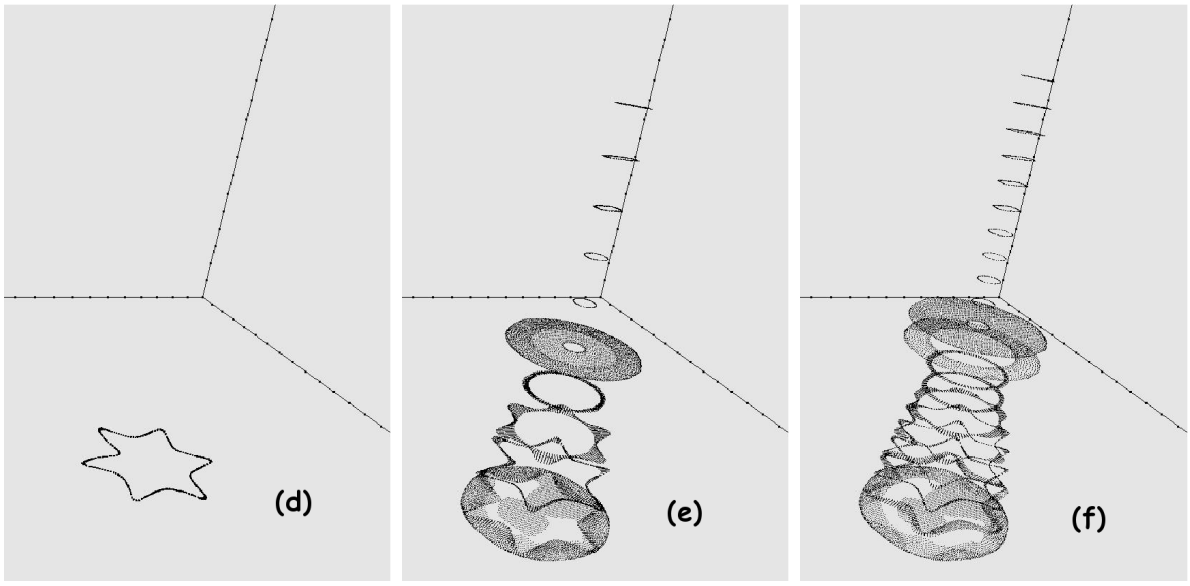


Figure 3.6: Instances of the sliced screwdriver with different values of thickness. A cross section with $1/4$ thickness projects to slice (d). The entire screwdriver sliced with a specific thickness (e), and half its thickness (f).

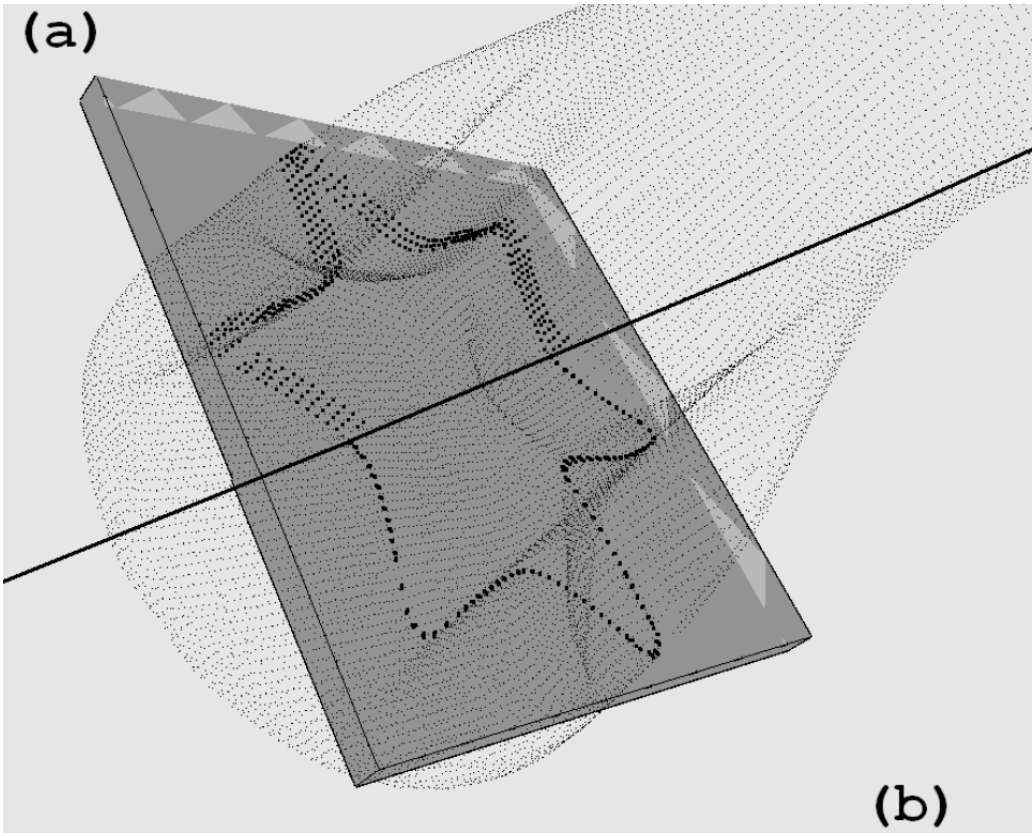


Figure 3.7: Detail of the Cyberware Screwdriver. (a) One cross section is highlighted, and (b) the points of the cross section are projected to its corresponding 2D plane.

In some cases, defining the principal axis may prove to be a complex procedure, like the one in figure 3.8, which illustrates a hip bone point cloud with 132538 points also from Cyberware [1]. The process of identifying the features of the model may prove to be a difficult task.

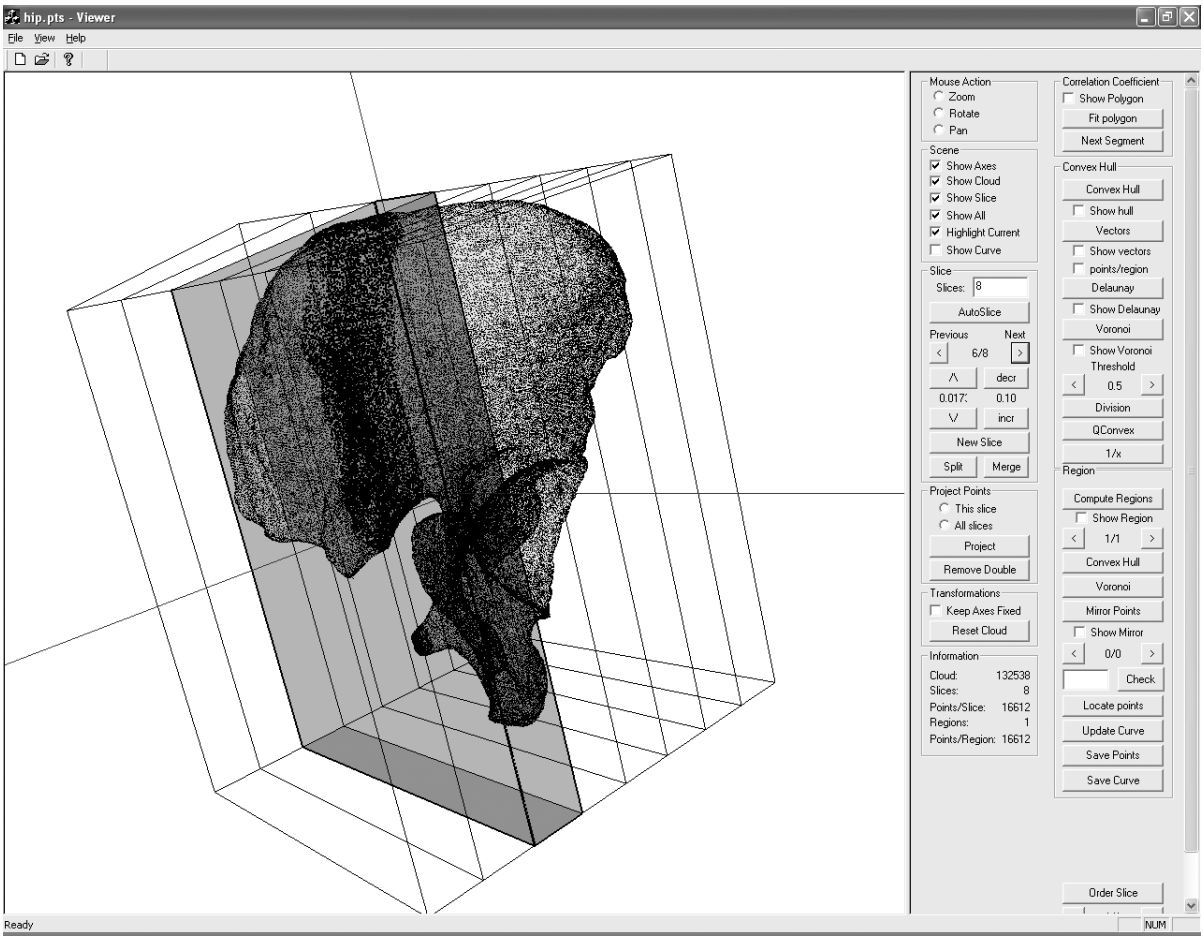


Figure 3.8: The Cyberware hip bone. The proper slicing direction is not obvious in this model, as the medial axis of the cloud is not a simple curve.

Figure 3.9 shows an example of a point cloud that represents a Cycladic idol, and the acquired point cloud, which consists of 131517 points and was acquired using a base Scanny 3d color laser scanner [156]. The Cycladic idol is one of the most recognizable specimens of ancient Greek sculpture. For visualization purposes, the point cloud in figure 3.9(d) has been sliced in 25 cross sections only. For the actual computations, which have produced the models of chapter 6, the point cloud was sliced into 100 cross sections.

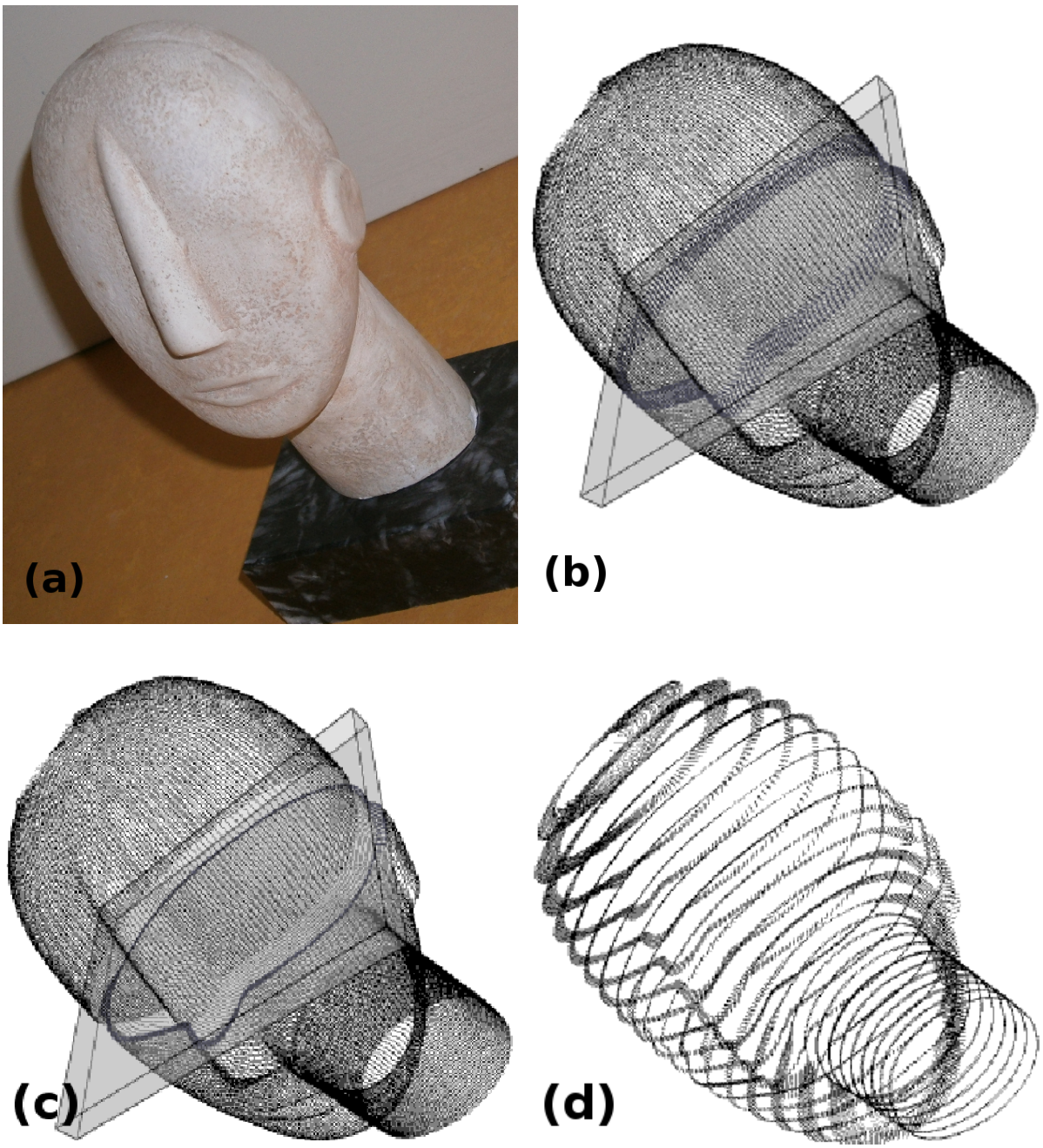


Figure 3.9: The Cycladic Idol. (a) The actual object, (b) the point cloud of its surface scan, with one cross section highlighted, (c) the cross section is projected to the 2D plane, (d) the entire point cloud is sliced into cross sections, and the points of each cross section are projected to its corresponding 2D plane.

Another example is the one of figure 3.10 which illustrates the scan of a twist drill bit. This point cloud consists of 1436231 points, and was also acquired using a base Scanny 3d color laser scanner [156]. Again for visualization purposes the point cloud in figure 3.10 (right) was sliced into 40 cross sections, while in the actual computations it was sliced into 200 cross sections.

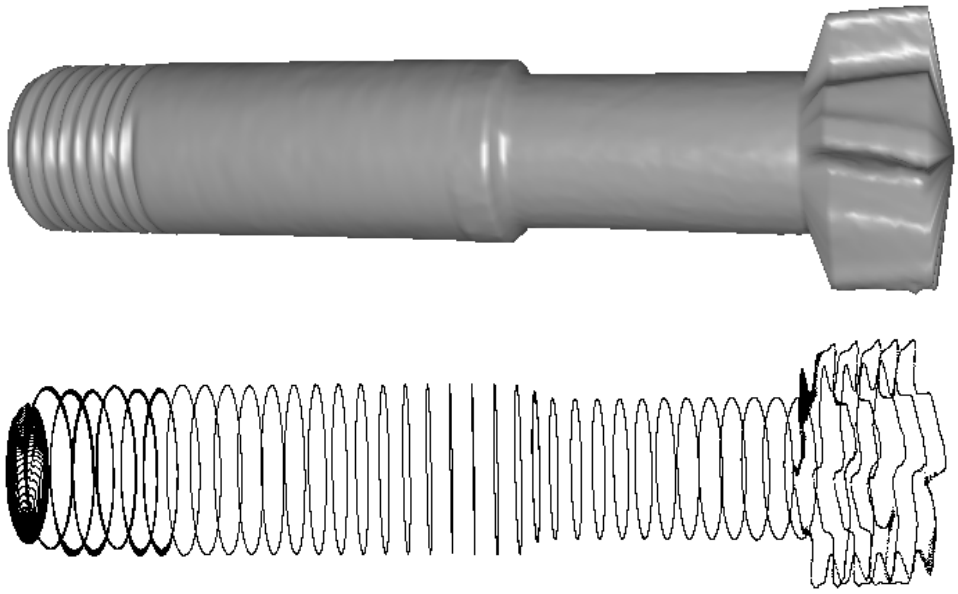


Figure 3.10: A twist drill bit. (Up) The actual model, (Down) The point cloud sliced into cross sections.

Chapter 4

Computation of the Feature Poly-Line

4.1 Introduction

4.2 Computation of the Feature Poly-Line

4.3 Computation of the Convex Hull

4.4 Ensuring proper region assignments

4.5 Computation of the Voronoi Diagram

4.6 Examples

4.7 Performance Analysis

4.1 Introduction

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 by the points.

Having computed the cross sections of the point cloud, we managed to provide some kind of organization to the point set, but we are still unable to perform editing of any form, as there is no information available concerning the connectivity between the points. What we need to do is to extract information about the boundary of the point set in each cross section. This boundary needs to be expressed in the form of an ordered point list that later on will be represented as a smooth continuous curve.

The ordered point list is referred to as the *Feature Poly-Line*, as the feature points form a polygon that is regarded as a sequence of parametric curves of degree one. When all cross sections are represented with parametric curves of higher degree, the user will have a smooth and flexible editable model at their disposal.

4.2 Computation of the Feature Poly-Line

To extract the information about the geometry of the model from the unstructured point cloud, we first have to represent the boundary of the cross sections as closed curves. At first we begin by computing a linear curve, i.e. an ordered point list called the feature poly-line. In the next step of our method, this feature poly-line will be replaced by a curve of higher degree, which will provide a smooth continuous representation of the model. But for the moment, we just need to identify some points of the cross section that describe the boundary accurately.

Initially, each cross section may consist of many points located close to each other, as provided by the point cloud. The information provided by so many points is unnecessary, so we only need to keep some representative points on the boundary of the cross section, i.e. an ordered point list consisting usually of much fewer vertices than the points of the cross section. This boundary representation, called the feature poly-line, is extracted with the use of computational geometry methods as described below.

4.3 Computation of the Convex Hull

In the beginning we compute the convex hull [157] of a cross section. The points of the convex hull are boundary points and are immediately identified as feature points, as illustrated in figure 4.2. The computations are performed using the Qhull library which is available in [157].

In most cases the shape implied by the points in a cross section is not convex, and the feature poly-line formed by the convex hull will not describe the cross section accurately in all regions. In such cases, fitting a poly-line to these points needs additional information beyond the convex hull. But there is a special case where

the shape implied by the points in a cross section is convex, and the feature points from the convex hull will form the feature poly-line, as it will accurately describe the boundary of the entire cross section.

For example, if the points of a cross section form a rectangular shape, the convex hull will consist of only four vertices and four edges, while the cross section may consist of thousands of points. Such an example is illustrated in figure 4.1.

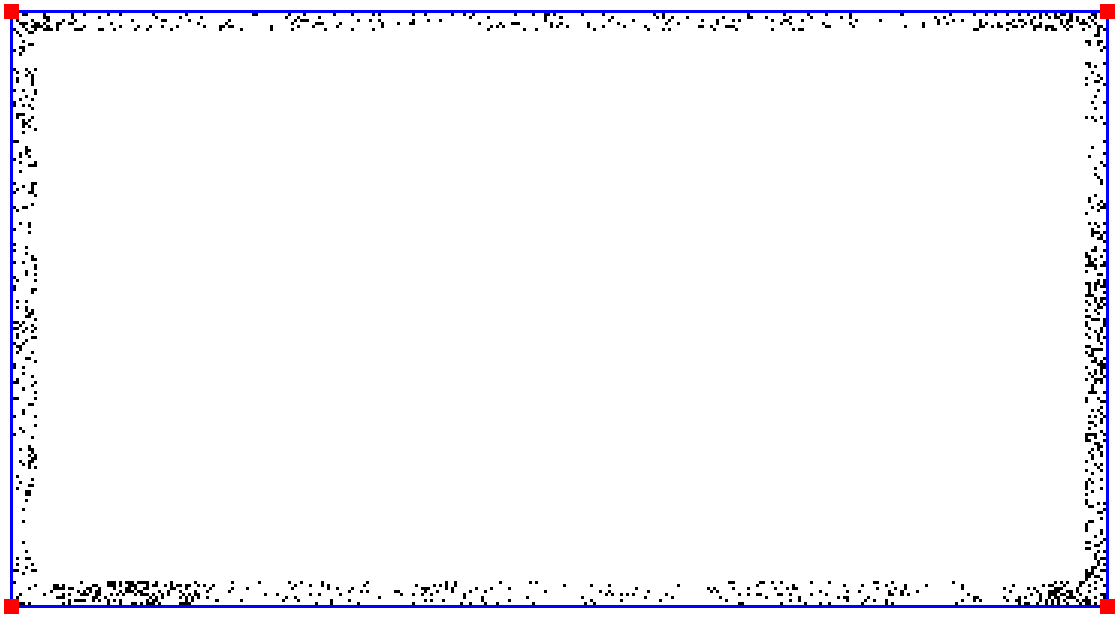


Figure 4.1: A point set with the shape of a rectangle is convex. The feature poly-line matches the ordered point list defined by the convex hull, as it describes the boundary sufficiently at all regions.

In this case, no more computations are required for this cross section. But in the general case of a free form model, there may be other concave regions, which are not properly described by the convex hull. In such cases, like the one illustrated in figure 4.2, computing the convex hull is only the first step of the process.

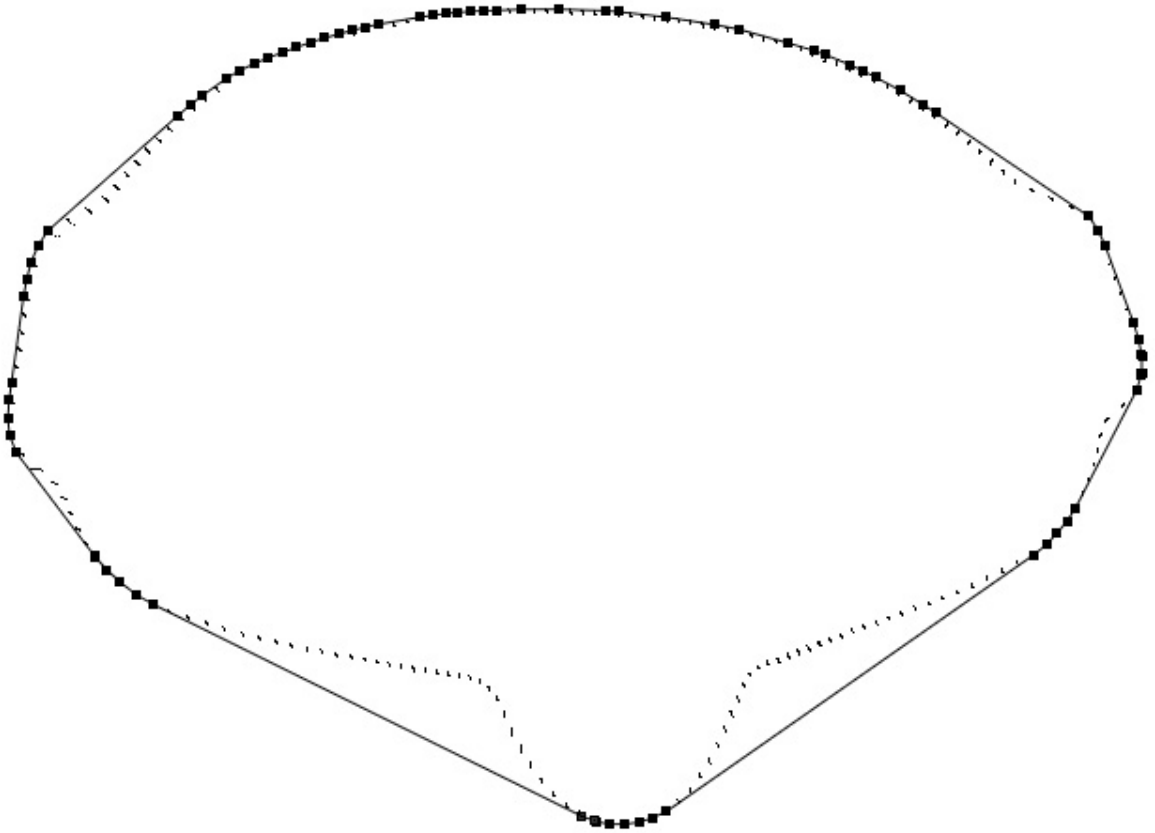


Figure 4.2: A cross section of the cycladean idol point cloud. The vertices of the convex hull are identified as feature points for this cross section. Some regions are not yet described by the feature poly-line.

For the general case in which we have regions not properly described by the convex hull, we define a set of regions on the cross section, one region for each line segment of the convex hull. Then we assign the points of the cross section to their corresponding regions, according to their euclidean distance from the convex hull. After we assign each point of the cross section to a region of the convex hull, we determine which regions need to be described more accurately. The average distance of the region points from the feature poly-line is a good estimator for determining whether the region is properly described. Now there are points that are located near the convex hull (up to a threshold), while other points are still located far from the convex hull. Some of these regions may consist exclusively of points very close to the convex hull (convex regions), while other regions contain points located far from it (concave regions), and need further processing.

4.4 Ensuring proper region assignments

Before computing the feature points for the concave regions, we need to ensure that all points of the cross section have been assigned to the proper region. There are some cases that might compromise the next computations and threaten the effectiveness of the interpolating poly-line. The problem arises when one or more points are assigned to the wrong region of the curve. One would expect that each point belongs to the region of the curve which is closest. But 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 feature poly-line (e.g. see figure 4.3).

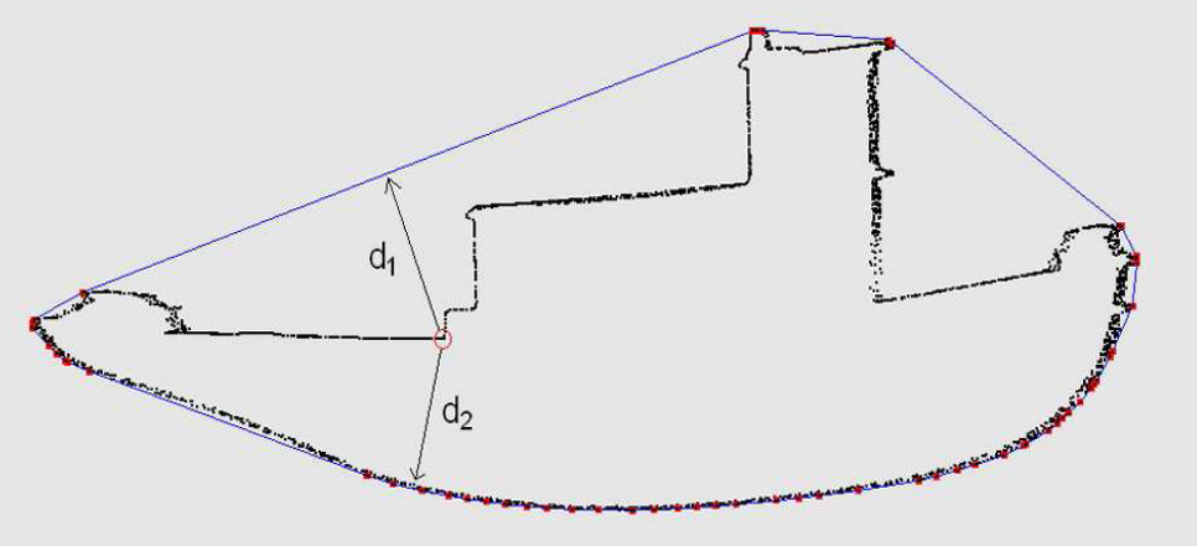


Figure 4.3: A cross section of the point cloud of the Cyberware boat [1]. The points in the highlighted area (small circle) are located closer to a region other than they should be assigned to ($d_2 < d_1$). We need to use the information of their neighboring points to assign them to the correct region.

To avoid having such erroneous assignments, we keep track of the previous point assignments, and if one point is found to be closer to one region, while its neighbors are closer to another, we ignore the distance to the closest region and assign the point to the region we assigned the neighboring points. When the distance is close to zero we can change region. To ensure each point of the cross section is assigned to its proper region, we declare the following definitions:

Definition 4.1. 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 4.2. We call separability tolerance $d > \epsilon$ 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 $\|p_1, p_2\| < d$.

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

Definition 4.3. The shortest neighbor path between two points s_1 and s_n of the point cloud slice is a sequence of points $[s_1, s_2, ..., s_n]$ such that each point is neighbor to the next and $\sum_{i=1}^{n-1} \|s_i - s_{i+1}\|$ is minimized.

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

Definition 4.4. The initial seed of a region r_i is the shortest path between p_i and $p_{((i+1) \mod n)+1}$.

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 4.5. 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.

If we have the above definitions in mind while assigning the points of a cross section to their corresponding regions, all points will be assigned to the proper region, even if it is not the closest.

4.5 Computation of the Voronoi Diagram

In the previous step, in which we started identifying feature points using the convex hull, all feature points have a common characteristic, i.e. all the feature points lie on the outer boundary of the projected points in the cross section. Since we are extracting a boundary representation of the the 3D object, the idea is to construct a feature poly-line which consists of external points only. Using points that are not on the external

boundary of the cross section would cause problems on the resulting model. These problems are discussed in section A.3 of the Appendix.

The convex hull may have described the boundary of the model in some regions, but there may be other concave regions where the convex hull does not capture the topology of the points accurately. In other words, in some regions the points lie very close to the convex hull and in some other regions the points lie far from it. Obviously the convex hull is not efficient in describing these regions and another approach has to be used.

To identify more feature points in regions where the curve is not close enough to the points of the slice, we use the Voronoi diagram, which has a property called the largest empty circle [158, 159]. 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 center of this circle is one of the Voronoi vertices of the region. Therefore, if we compute the Voronoi diagram for a specific region, each Voronoi vertex is a center for a largest empty circle that is touching three or more slice points. The case of more than three points touching a circle appears when we have many points (more than three) with the same radius from the center, i.e. the Voronoi vertex. In this special case all points share the same Voronoi center.

We can use this property on a region that consists of points which are located far from the curve, to identify external points of the region that would qualify as additional feature points in the poly-line. The three (or more) points of contact can be added to the set of candidate points for the feature poly-line, as they are external points of the region, which could be used to update the curve to describe the region more properly than the convex hull. We can ensure that these points are external points, by using largest empty circles thrown towards the point cloud from the outside of the model. An example is illustrated in figure 4.4.

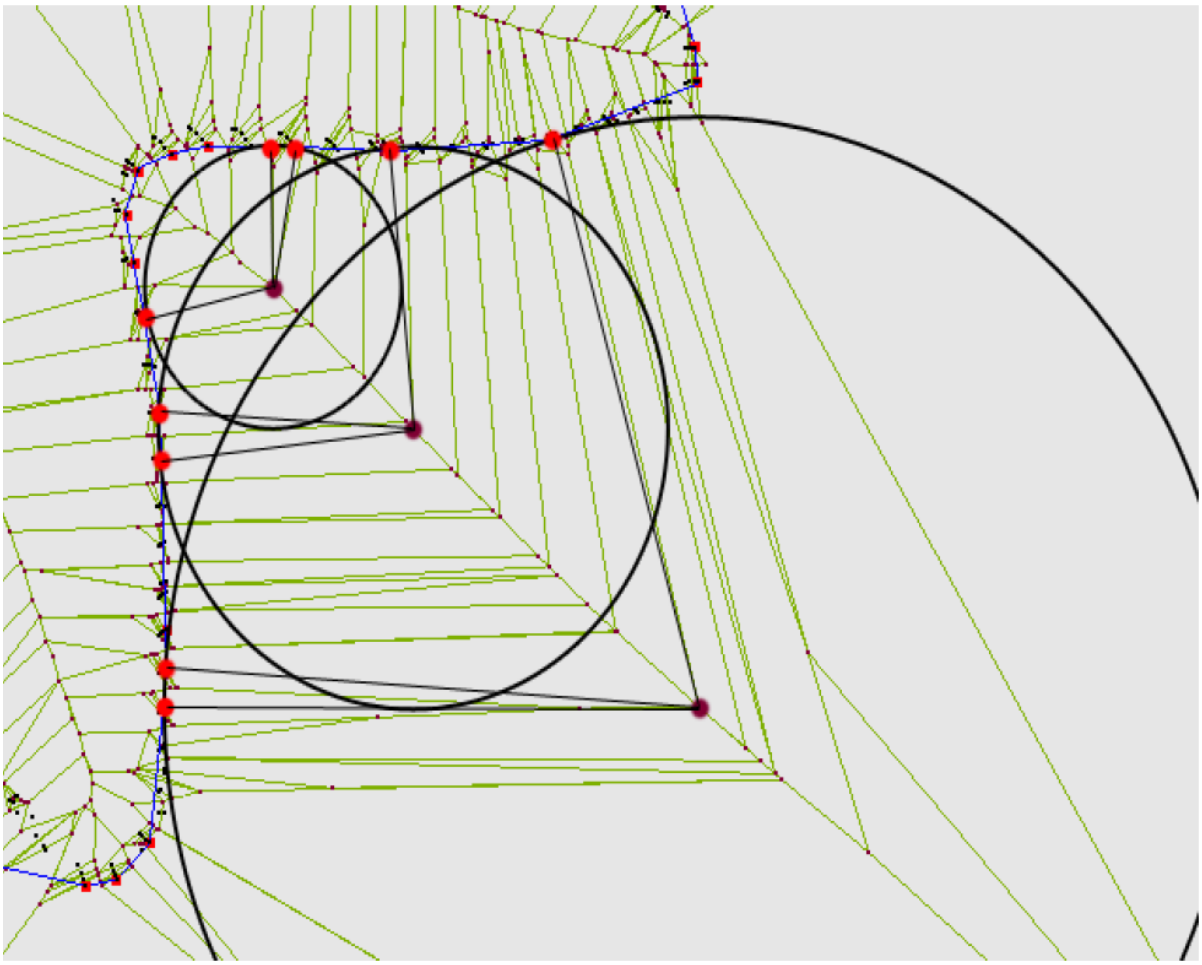


Figure 4.4: Detail of a cross section of the handle of the screwdriver point cloud. The largest empty circle of a Voronoi vertex is used to identify additional feature points and update the curve.

The use of the Voronoi diagram and the largest empty circle for identifying feature points has also been used in the rotating ball technique [86]. The identified feature points also form the so called alpha shape of the point set, as described in [87].

The computation of the Voronoi diagram is performed using the Qhull library [157], as was the Convex hull in the previous step of the method. In this step, we compute the Voronoi diagram for those regions only, which need further processing, i.e. the regions that have points far from the poly-line. Each Voronoi vertex located outside the region is a candidate center for a largest empty circle that is touching three or more region points. Of course the Voronoi diagram consists of other vertices that lie between the points of the region, or in the inner side of the model.

We are interested in these centers only, which are associated to points of the cross section that lie on the external boundary of the model. So 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 inner side of the point cloud are also being excluded, because we are interested in feature points on the external 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.

In practice, after we have eliminated all unsuitable Voronoi vertices, we end up with a subset of centers, which can be used to identify external points on the region. The user may then request to use as many of these centers as they need, according to shortest distance from a set of reference points which are evenly dispersed on the outside of the region. This will ensure that we will not use centers very close to the region, very far from the region, very close to each other, or on the inside of the region. The number of centers may even be chosen arbitrarily. If the number of centers is small, an additional iteration may be required to refine a part of the region. A large number of centers will just identify more feature points, which can be discarded later if they cause problems, such as the over fitting effect we will discuss in section 5.3. Figure 4.5 illustrates an example of the centers which are chosen for a region.

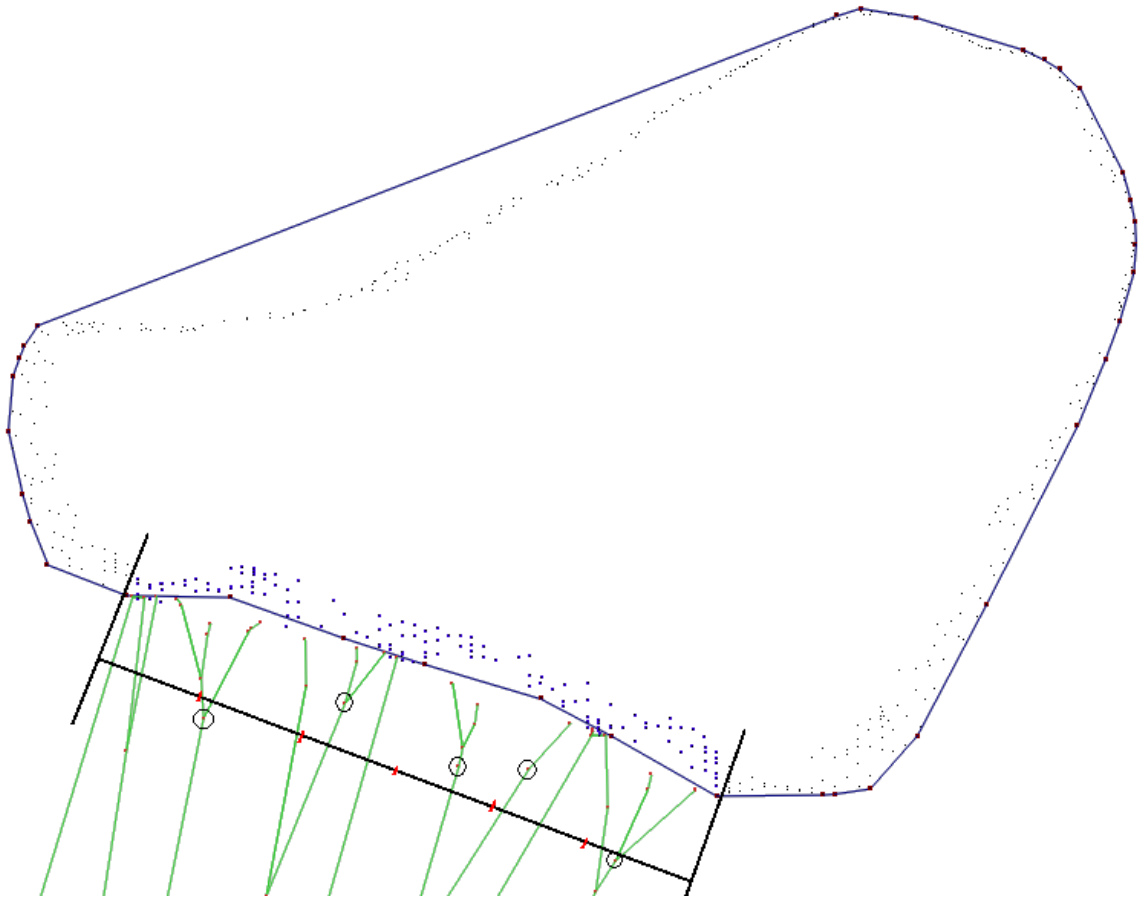


Figure 4.5: The user requests the number of centers to be used for identifying additional feature points. The centers closest to the reference points defined at equal intervals on the outside of the feature poly-line are used.

After choosing a fair number of Voronoi vertices, we locate the three (or more) points that are associated to each of them, i.e. they lie on the largest empty circle for each o these centers. This is equivalent to choosing the points of the unique Delaunay triangle, considering that there aren't more than three co-circular points for this Voronoi vertex (if there are more, the triangle is not unique, but still we can choose any of the points, or even all of them). We identify these new points as feature points and update the curve accordingly. We repeat the process for all regions that consist of points which are far from the feature poly-line, until all points are located near the curve. The idea is similar to the iterative multigrid methods in numerical analysis, which are based on a sequence of meshes obtained by successive refinement. Such algorithms have a recursive structure, i.e. in each multigrid iteration the error is smoothed at a certain grid, the residual is transfered to the next coarser grid, and in the next iteration the grid with the coarser residual is corrected (recursively) [160].

Multigrid methods are typically used for accelerated convergence in adaptive mesh refinement.

As we discuss in section A.4 of the Appendix, there are many candidate Voronoi centers that may be used for identifying additional feature points. But there are some issues that make some of these centers inappropriate, as they would result in models of lower quality. For example, by choosing the farthest Voronoi vertex every time, the feature points are expected to be relatively close to each other, as the curve closes in to the region points slowly. We can ignore some Voronoi vertices that are too far from the region, and choose a Voronoi vertex that is the farthest within a bounded area. We can choose e.g. a Voronoi vertex that is farther from the region, but not farther than the maximum distance of the region points from the curve. This condition is indicative for the speed of the curve convergence. If we do not restrict the Voronoi vertices it will take many steps to fully update the curve on this region. The curve will consist of more feature points, and it will describe the region points with increased detail.

On the other hand, if we restrict the Voronoi vertices within a radius, it will take fewer steps to update the curve, the feature points will be fewer, but the curve would describe the region points less accurately. We use this parameter to adapt the curve fitting according to user specifications requirement or quality of approximation guaranties. Figure 4.4 illustrates the resulting curve in case we choose the farthest Voronoi vertex within a restricted area each time. If we had chosen the farthest Voronoi vertex in each step, the resulting curve would consist of more feature points, the curve would describe the slice points more accurately, but more iterations would be required to fully update the curve.

By combining the initial convex hull with the feature points provided by the selected Voronoi vertices of each region, we get a feature poly-line 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 poly-line. After each iteration, the feature poly-line is updated with the newly identified feature points, and the regions are recalculated, so that each region of the feature poly-line has its own set of slice points (In the beginning we had regions for the convex hull, but now the feature poly-line is used for calculating the regions). The final result of the fitting feature poly-line is illustrated in figure 4.8.

The method is summarized in Algorithm 4.1. Note that L is a plane vertical to

the slicing axis.

Algorithm 4.1 The algorithm for the feature point extraction using the Voronoi diagram.

Voronoi_feature_poly-line()

Input: a set P of points, Slice i

Output: an ordered set F_i of feature points

$(P_i^{(3D)}, L) \leftarrow$ slice(i, P)
$P_i \leftarrow project(P_i^{(3D)}, S)$
$F_i \leftarrow qconvex(P_i)$
$F_{ij} \leftarrow \emptyset$
repeat
  for each region P_{ij} of F_i **do**
    if $avg\_dist(P_{ij}, F_{ij}) > \epsilon$ then **then**
      $V_i \leftarrow qvoronoi(P_{ij})$
      $V_{candidate} \leftarrow V_i - excludedVoronoivertices$
      $F_{ij} \leftarrow largest\_empty\_circle(V_{candidate}, P_{ij})$
      $F_i \leftarrow F_i \cup F_{ij}$
    end if
  end for
until $F_{ij} \neq \emptyset$
return F_i

In the implementation of our method, we have defined a maximum number of allowed regions, and automatically process the larger regions, as they are most likely the regions which need further processing. The iterations go on until we have reached the maximum number of regions. However, the user is also allowed to specify which regions should be processed further, and may also manually select or remove Voronoi vertices, region points or feature points in case the results are erroneous (e.g. because of noisy data).

Of course, each time the feature poly-line is updated with additional points, the regions are recalculated accordingly, so that the points are once again assigned to their corresponding regions. The entire process is repeated as long as there are points that are not close to their regions.

4.6 Examples

Figure 4.6 illustrates an example of a region, in which the topology was not properly described by the feature poly-line after computing the convex hull. The Voronoi diagram is computed for this region, and the feature poly-line is updated with additional feature points, according to the property of the Voronoi diagram, the largest empty circle.

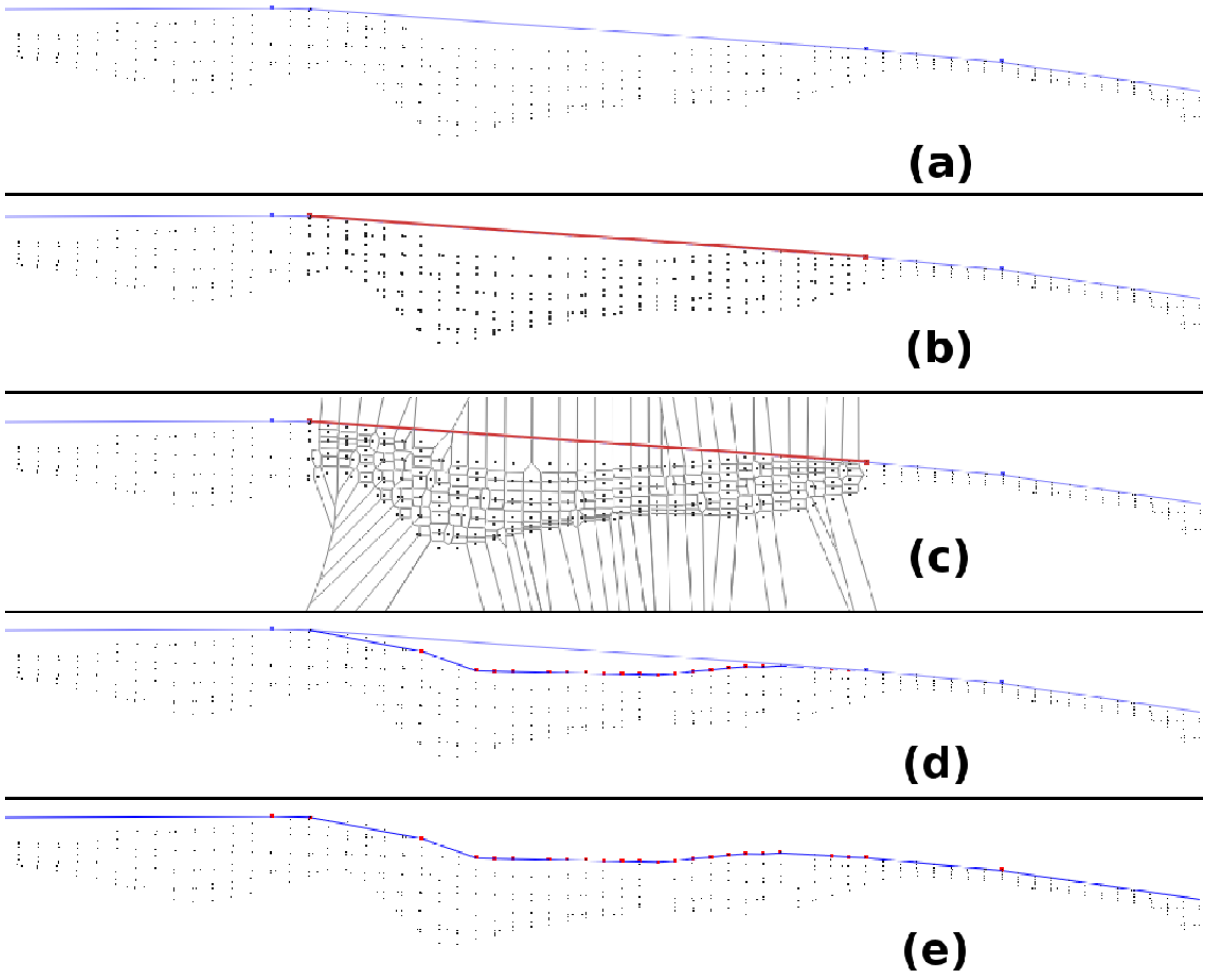


Figure 4.6: Detail of a cross section of the cycladean idol point cloud. (a) The convex hull provides the initial feature poly-line. (b) Some regions are not properly described. (c) The Voronoi diagram is computed for those regions. (d) Additional feature points are identified. (e) The feature poly-line is updated.

The next example (figure 4.7) shows an entire cross section, in which we can see which regions need further processing. After processing the large region on the first image, a small region is still not properly described. We process this small region again, and repeat until all regions are properly described.

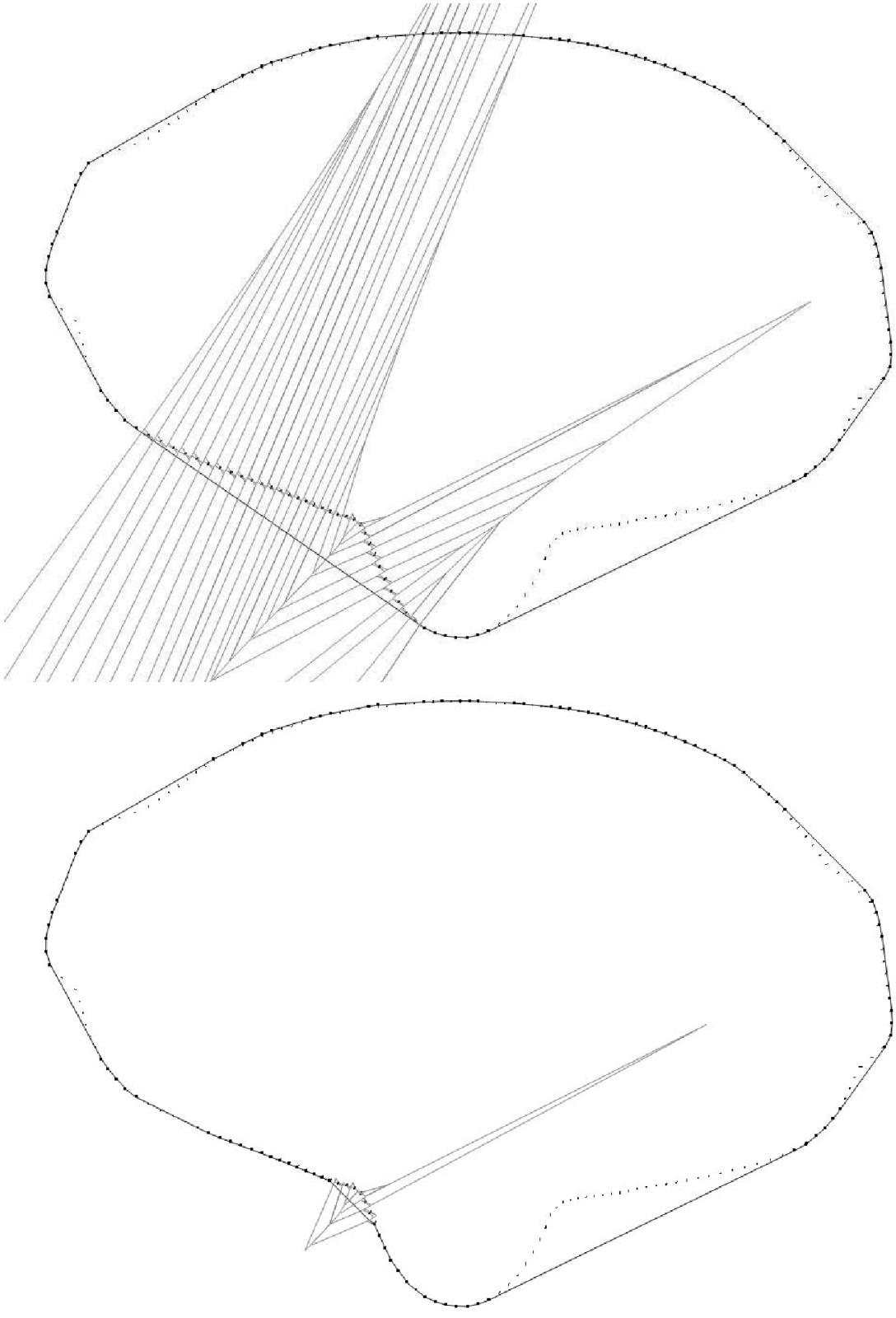


Figure 4.7: A cross section of the cycladean idol point cloud. For the feature polyline to fully describe a region, several iterations may be required. (Top) The first iteration of our method applied on a region. (Bottom) The second iteration for the small region.

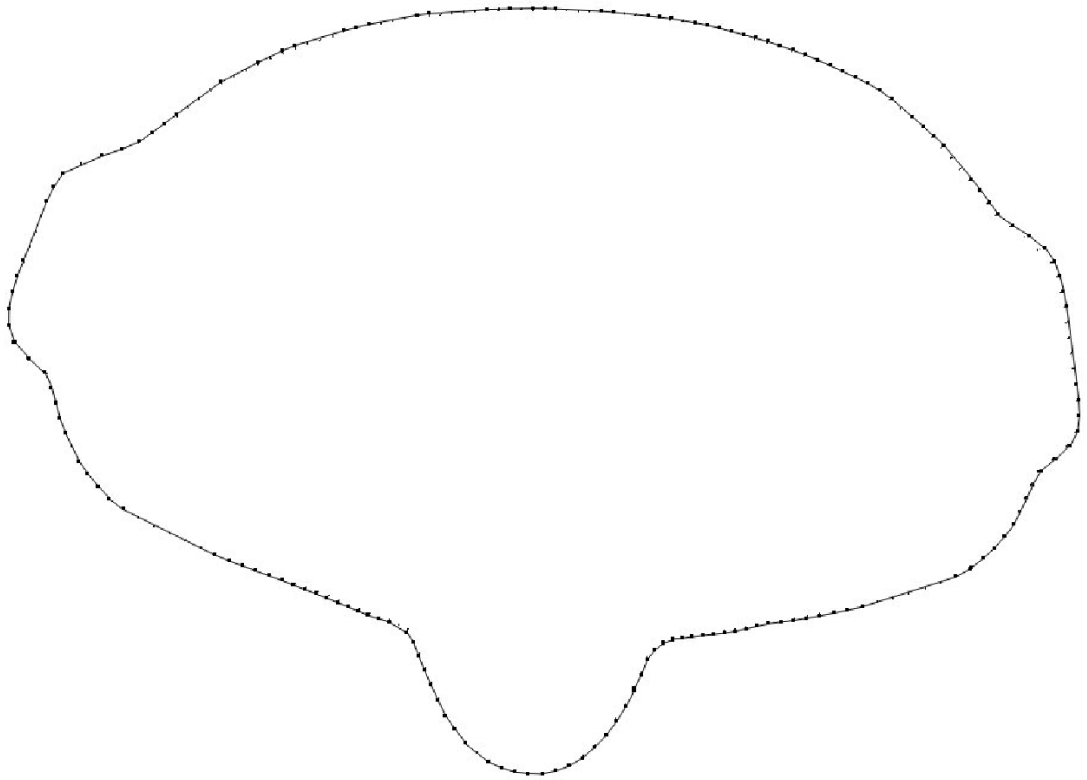


Figure 4.8: A cross section of the cycladean idol point cloud (cont.). After the second iteration the region is described as expected. All other regions have also been processed.

Figure 4.9 shows one cross section of the twist drill bit in figure 3.10. The feature poly-line 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 poly-line describes the entire cross section accurately.

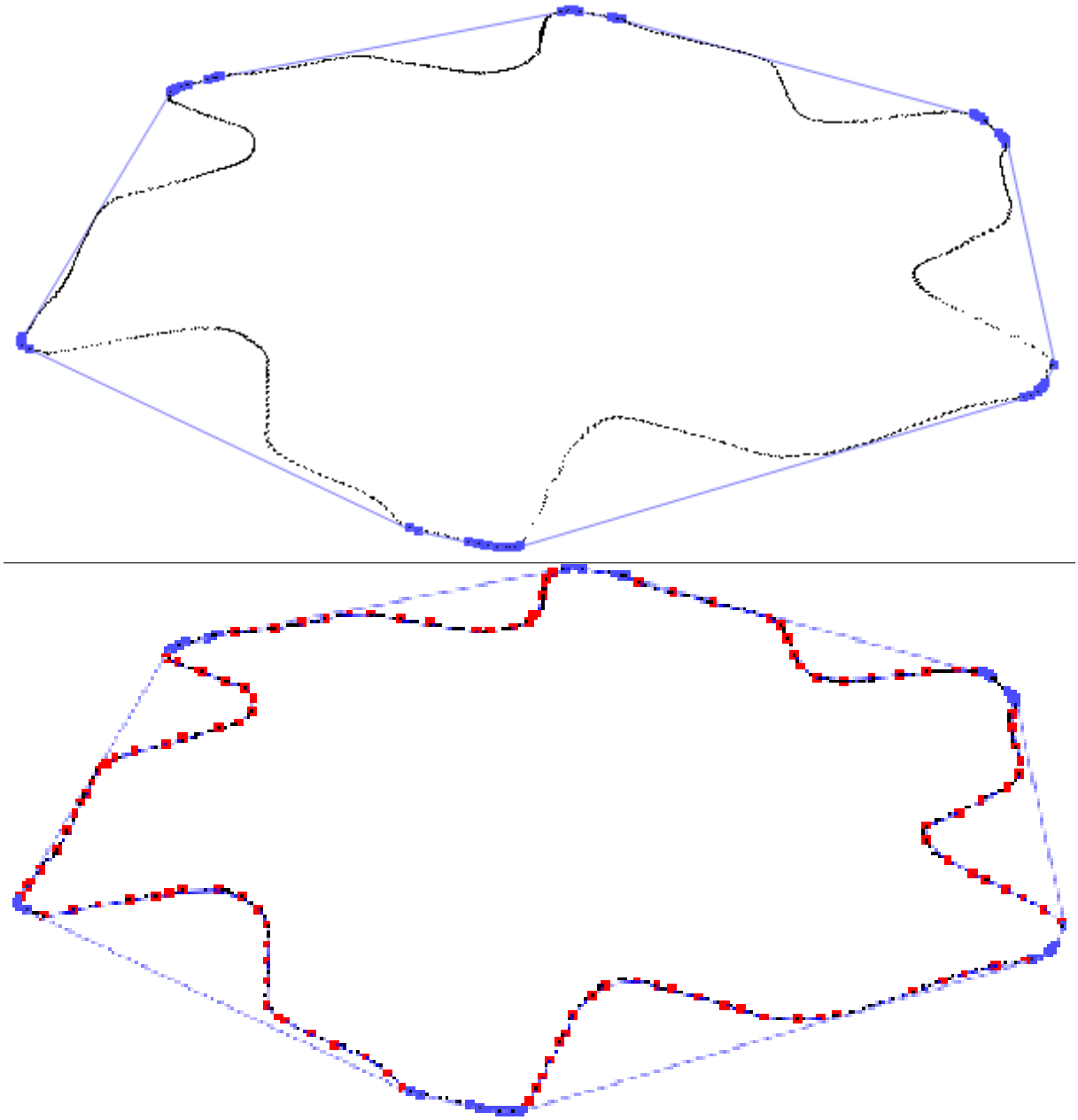


Figure 4.9: A cross section of the twist drill bit point cloud. The points of a cross section are processed as an individual 2D point set. (Top) The convex hull of the point set (blue points) provides the initial feature poly-line. (Bottom) The feature poly-line is updated as new feature points are added in regions where the point set is not accurately described (red points).

4.7 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. In this section we provide an estimation of the expected complexity as a function of 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 their 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 poly-line. 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 $\sum_{i=1}^{s} \sum_{j=1}^{r} n_{ij} = n$.

The number of iterations required to fully fit the feature poly-line to the slice points also depends on the shape of the points. 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. In the example of the screwdriver handle cross section in figure A.7, the resulting curve is satisfactory after the second step. In the cross section of the Cycladean Idol in figure 4.7, the resulting curve is also satisfactory after the second step. However, the twist drill bit cross section in figure

4.9 requires at least five steps for all the large regions to be properly described by the feature poly-line.

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 poly-line it requires a constant number of point operations.

To fully update the feature poly-line 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. This is bounded by $O(\log n) \cdot O(n \log n) = O(n \log^2 n)$. So, in conclusion, to derive descriptive feature point sets for all slices of the point cloud takes $O(n \log^2 n)$ time.

Chapter 5

Reconstruction of the Surface

5.1 Introduction

5.2 Inserting new Cross-Sections

5.3 Representing Cross-Section Contours by G^1 Splines

5.4 Reconstructing the Surface

5.1 Introduction

Up to this point, we have computed a feature poly-line for each cross section of the point cloud, which represents the boundary of the cross section accurately. However, this is not the best we can do, as the set of poly-lines is linear and the representation will not be smooth. We need to provide a smooth continuous representation of these feature poly-lines, and use it to reconstruct the surface of the model.

But before we proceed in constructing this smooth continuous representation, there is another issue we should discuss, which should be addressed at this point, where we still have our model in the form of a set of feature poly-lines.

Among the cross sections we have computed, there may be some parts where the representation is accurate in each 2D cross section, but the combination of the cross sections that will form the 3D surface may be erroneous. In other words, the cross sections we have computed so far may describe the point cloud accurately within a specific area, but there may be some areas between the existing cross sections where the point cloud is not properly described. Or there might be some cases where the shape of the feature poly-lines differs significantly between two adjacent cross sections.

Such an example is illustrated in figure 5.4, in which we can see two cross sections from the cycladean idol, at the part where the chin begins. The cross section on the top contains a protrusion at the area of the chin, while the bottom cross section is located under the chin, and therefore has no such protrusion. When we attempt to reconstruct the surface of the model, such major differences may be difficult to represent. For example if we construct a triangle mesh we will have very long triangles which may not connect to their neighboring triangles properly (the normals are abruptly influenced by the long triangles). Or if we use smooth patches for the reconstruction, the smoothness or the continuity between adjacent patches may not be preserved as we intend.

What we need is to reduce the differences between the feature poly-lines of adjacent cross sections. To achieve this, the first thought would be to divide the point cloud into more cross sections initially, so as to increase the level of detail among the cross sections. But this would mean we have to start the whole process from the beginning. Instead of discarding all the work we have done so far, we have included an option in our system that allows the user to artificially define a new intermediate cross section between two already processed cross sections, using only the existing feature poly-lines of the two cross sections.

5.2 Inserting new Cross-Sections

Again, we make use of the Voronoi diagram and the property of the largest empty circle. We compute the Voronoi diagram for all the feature points of both cross sections, and use the Voronoi vertices as feature points of a new intermediate cross section. To be precise, not all Voronoi vertices would qualify as feature points for the intermediate cross section, so we do not need to compute the entire Voronoi diagram of the two feature poly-lines. 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. Such points are illustrated in figure 5.1.

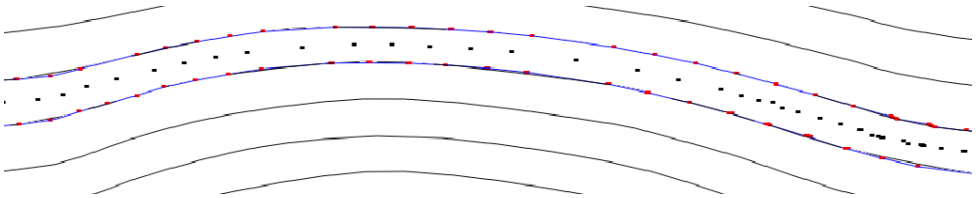


Figure 5.1: In case a new intermediate cross section is required, it is constructed from the Voronoi centers of the points of two adjacent feature poly-lines.

The process of matching the feature points of the two cross sections requires the triangulation of the contours defined by the two feature poly-lines. The correspondence between vertices of the two feature poly-lines is represented in a graph like the one illustrated in figure 5.2, which was discussed by Sederberg et. al. in [161]. 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 shortest distance between the points of the two cross sections as the cost function. Other popular approaches use different cost functions, e.g. to maximize the volume of the polyhedron that is formed by the triangle strip, as suggested by Keppel [138], or to minimize the surface area, as proposed by Fuchs [137].

All these works share a concept similar to dynamic time warping (DTW), a well-known technique used to find an optimal alignment between two given (time-dependent) sequences under certain restrictions. Originally, DTW has been used to compare different speech patterns in automatic speech recognition, and also in fields such as data mining and information retrieval, where it has been successfully applied to automatically cope with time deformations and different speeds associated with time-dependent data [162].

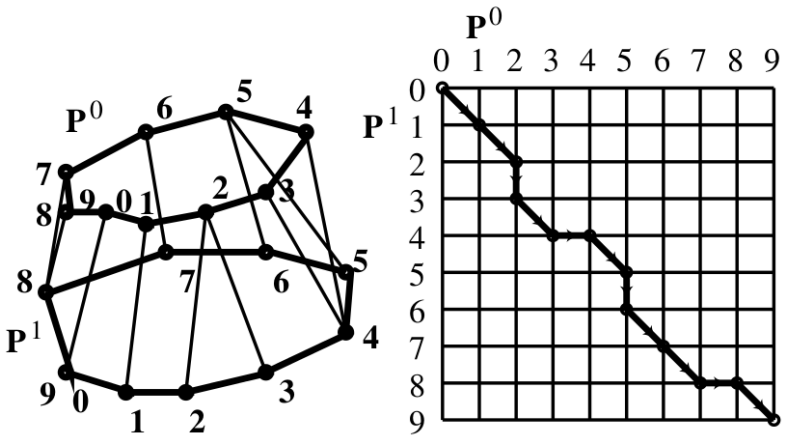


Figure 5.2: For the contour triangulation, the connectivity of the points is expressed as a monotonically decreasing path in a tabular scheme that specifies which points participate in each triangle or quad.

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. As a criterion for finding the proper correspondence between the points of the two cross sections, we have used the shortest Euclidean distance between the points of the cross sections. At the end, all Voronoi vertices for all feature points on both cross sections have been computed. Algorithm 5.1 describes the process of triangulating the contours of two adjacent cross sections to compute the feature points of the new intermediate cross section.

Algorithm 5.1 The algorithm for the contour triangulation.

contour_triangulation()

Input: point set P (n points), point set Q (m points)

Output: point set V of centers ($max(n, m)$ points)

$p_0 \leftarrow first\_point\_of\_P$, $q_0 \leftarrow find\_closest(Q, p_0)$

$compute\_path(0, n-1, 0, m-1)$ // compute path recursively

// compute triangulation according to path

for each p_i, q_j of P, Q, with $path\_table_{ij} = 1$ **do**

  if $path\_table_{(i+1)j} = 1$ **then**

    $triangle \leftarrow (p_i, q_j, p_{i+1})$

  else if $path\_table_{i(j+1)} = 1$ **then**

    $triangle \leftarrow (p_i, q_j, q_{j+1})$

  else if $path\_table_{(i+1)(j+1)} = 1$ **then**

    $triangle1 \leftarrow (p_i, q_j, q_{j+1})$, $triangle2 \leftarrow (p_i, q_j, p_{i+1})$

    // choose triangle with larger angles

    if $min\_angle(triangle1) > min\_angle(triangle2)$ **then**

      $triangle \leftarrow triangle1$ **else** $triangle \leftarrow triangle2$

    end if

  end if

  $center \leftarrow circumcenter(triangle)$

  $add\_point(\mathrm{V}, center)$

end for

Algorithm 5.1 The algorithm for the contour triangulation.

$compute\_path(start_1, end_1, start_2, end_2)$
if $end_1 < start_1$ **then**
  $return$
end if
if $end_2 < start_2$ **then**
  $return$
end if
if $end_1 - start_1 > end_2 - start_2$ **then**
  $middle \leftarrow (start_1 + end_1)/2$
  $nearest \leftarrow find\_closest(Q, middle)$

  $path\_table_{middle,nearest} \leftarrow 1$
  $compute\_path(start_1, middle - 1, start_2, nearest - 1)$
  $compute\_path(middle + 1, end_1, nearest + 1, end_2)$
else
  // $end_1 - start_1 < end_2 - start_2$
  $middle \leftarrow (start_2 + end_2)/2$
  $nearest \leftarrow find\_closest(P, middle)$

  $path\_table_{nearest,middle} \leftarrow 1$
  $compute\_path(start_1, nearest - 1, start_2, middle - 1)$
  $compute\_path(nearest + 1, end_1, middle + 1, end_2)$
end if

Figure 5.3 illustrates the steps of the procedure described in algorithm 5.1, in a visual representation. It follows a similar notation as the graph on figure 5.2. In the example of figure 5.3, a given point set P which consists of 20 points, and another point set Q of 17 points are used to compute the contour triangulation between the two cross sections P and Q.

The ordered point list P is shown in the bottom of the grid, and forms the horizontal axis of the tabular scheme. The point list Q is shown on the left side of the grid and forms the vertical axis of the tabular scheme. The cells $[i, j]$ of the grid designate the correspondence of the points p_i and p_{i+1} to the points q_j and q_{j+1}. In

practice, there are triangles in the contour triangulation which consist of these points.

In the beginning, the points p_0 and q_0 are assumed to be adjacent, since the two point sets are ordered and the starting points are the closest point in a given point of reference. Since the two point sets form closed poly-lines, the same applies for the last two points p_{20} and q_{17}, as they are the points just before the first points on both point sets.

First the middle point is chosen in the point set P, i.e. p_{10}, as P has more points than Q. The point that satisfies the given criteria (here being the shortest distance) is identified on the point set Q. As shown in the example of figure 5.3 (c), the point q_{11} is the closest to p_{10}.

The process is recursive, and the grids highlighted in gray on the top right and bottom left of the initial grid are used for the next iterations, until all points are combined to form the proper contour triangulation.

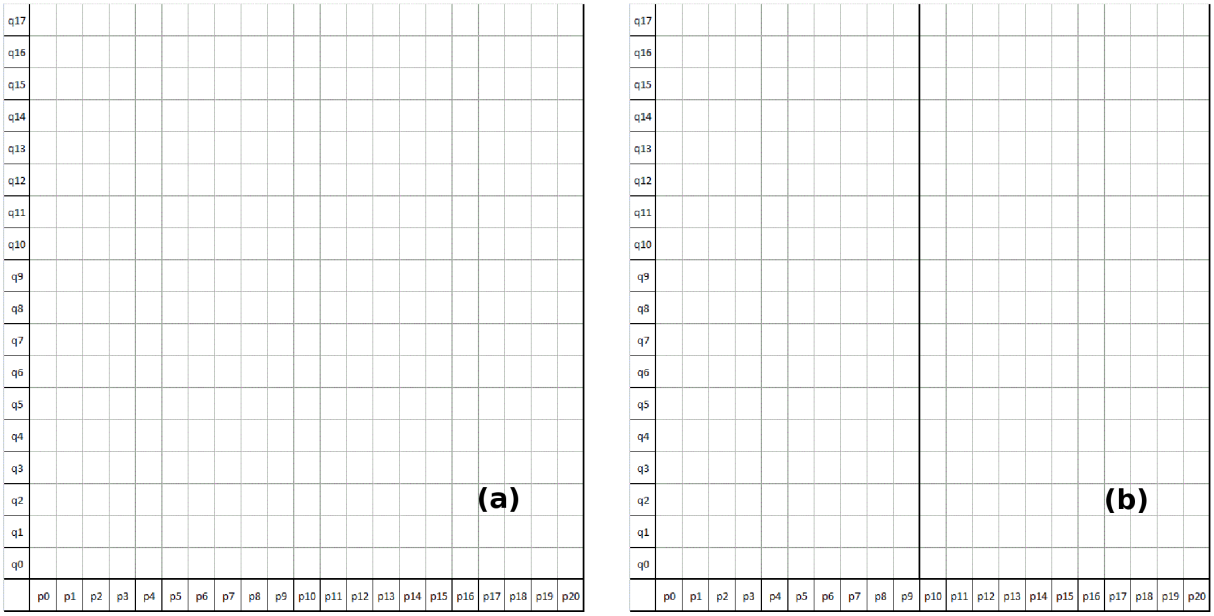


Figure 5.3: The recursive process of the contour triangulation. (a) The points p_i of the one cross section are positioned as a point list on axis x, while the points q_i of the other cross section are arranged as an ordered point list on axis y. The process includes: (b) choosing the middle point on the cross section with the greater number of points, ...

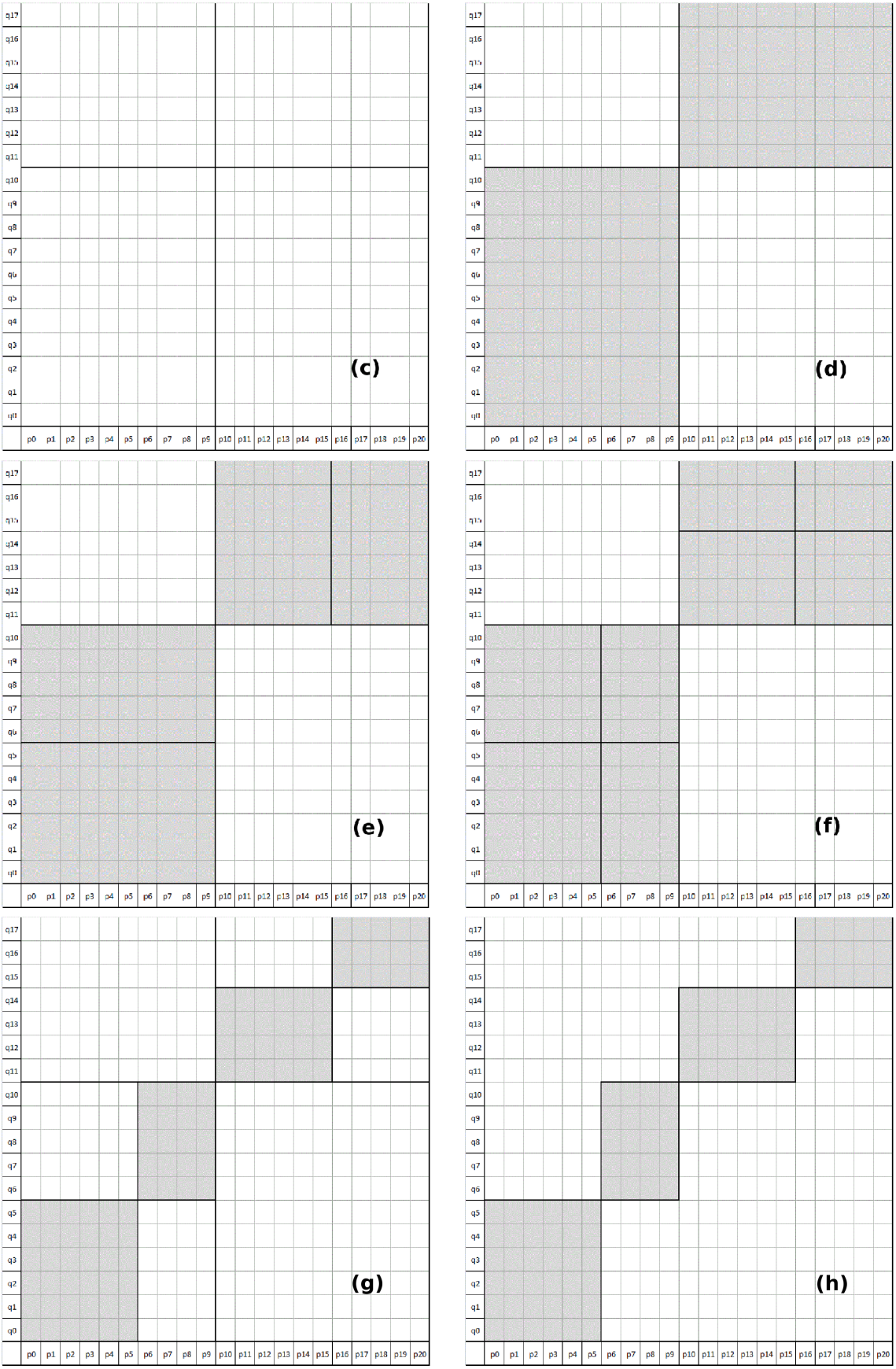


Figure 5.3: ... (c) finding its closest neighbor on the other cross section, (d) defining a correspondence between the two points, and (e, f, ...) repeating for the point lists on the left and right side recursively.

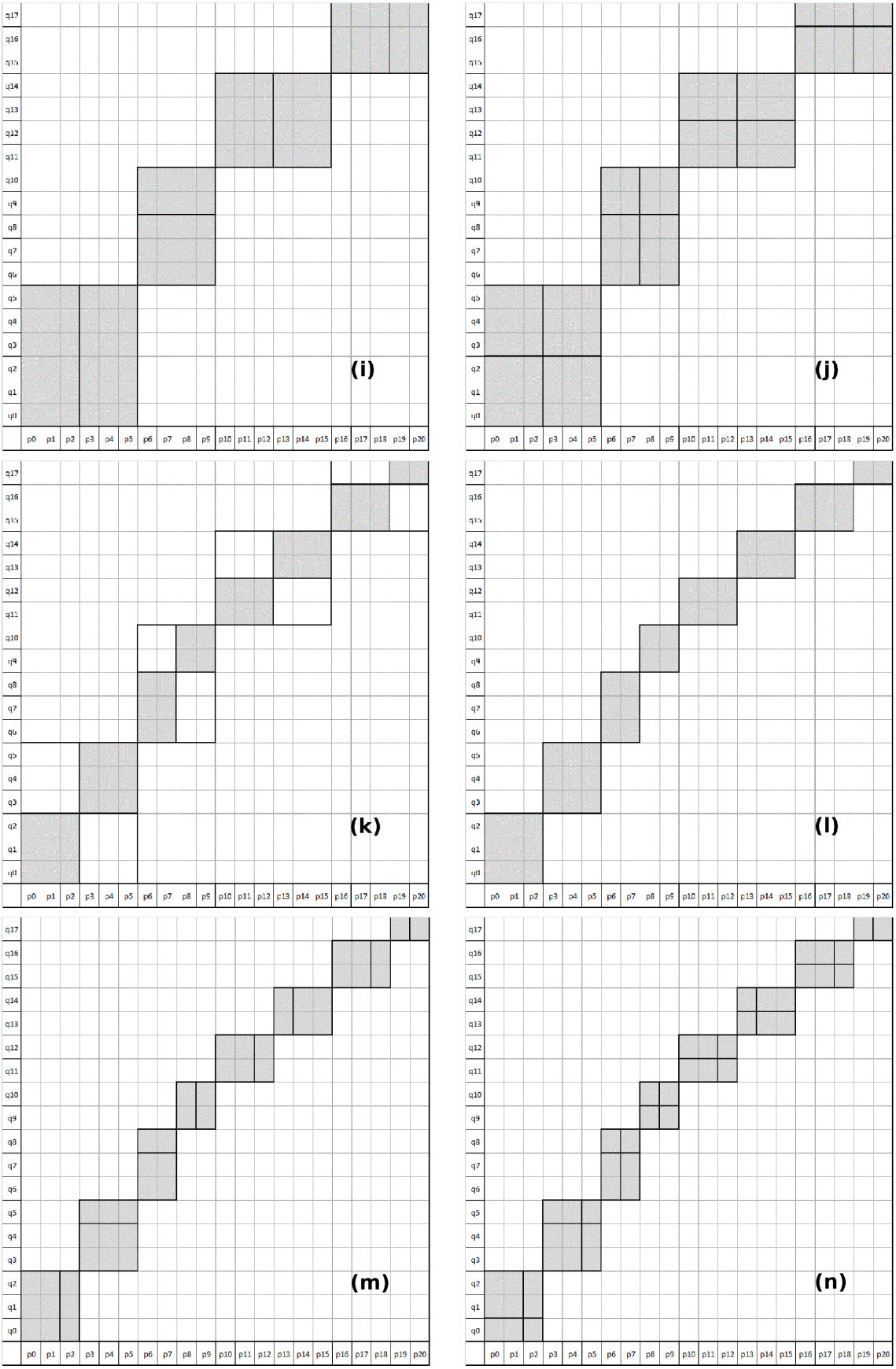


Figure 5.3: The recursive process of the contour triangulation (cont.).

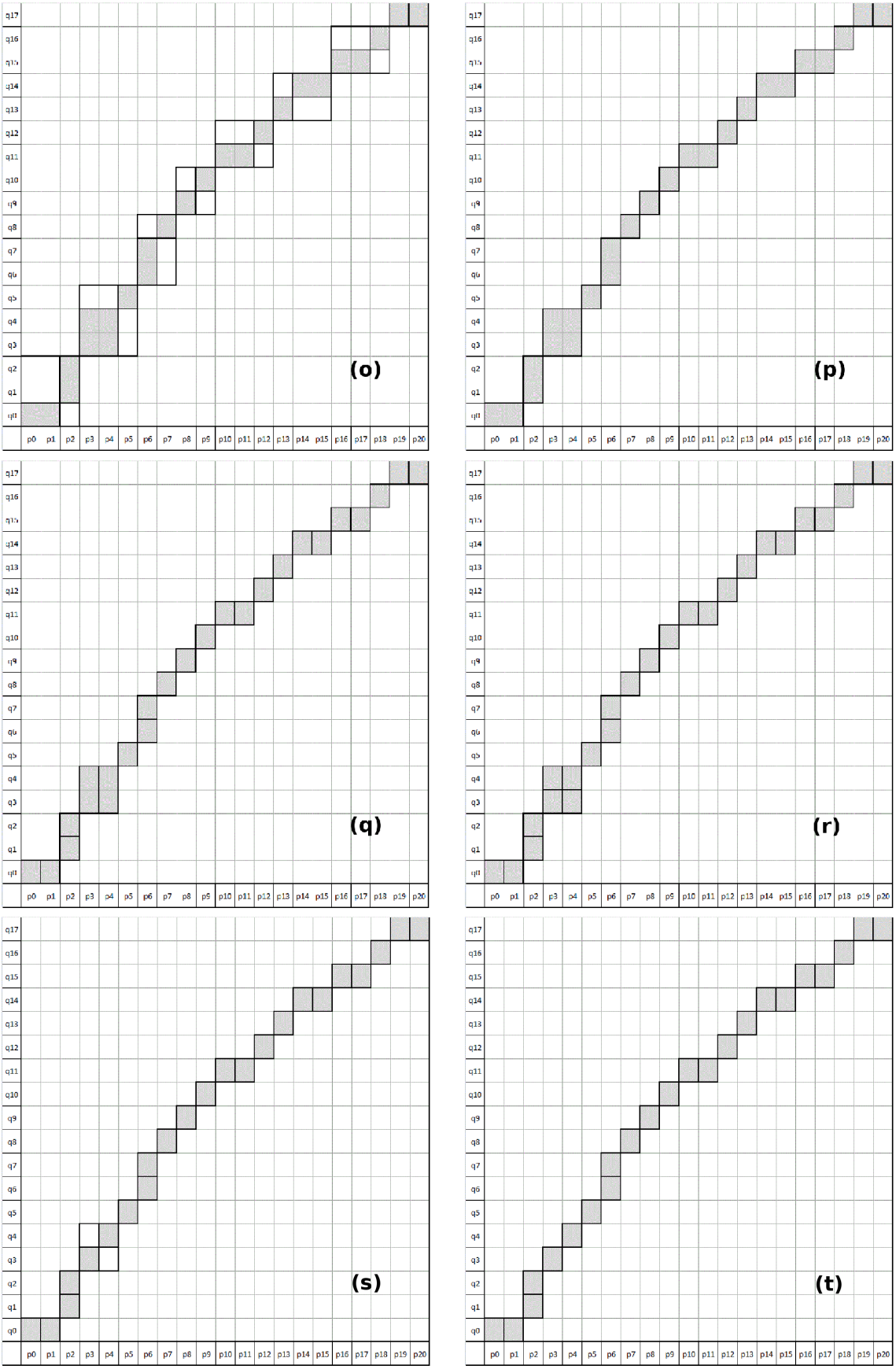


Figure 5.3: The recursive process of the contour triangulation (cont.).

The output of this method is a set of Voronoi vertices which will form the feature poly-line of the new intermediate cross section. These vertices are not positioned on a 2D plane, but in 3D space, so we need to project them to the corresponding plane of the new cross section, as we did with the cross sections of the point cloud.

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 the existing cross sections. Other information, such as cloud points or the convex hull is not available for this intermediate cross section, as the feature poly-line is computed by other means. However, there is no problem with that, since we only use the feature poly-line for further processing. Figure 5.4 shows a new cross section defined by two adjacent cross sections.

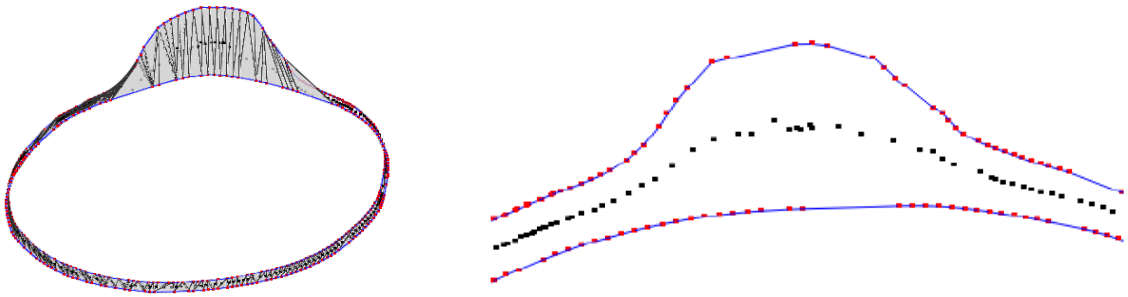


Figure 5.4: Cross sections in the area under the chin of the cycladean idol point cloud. (Left) A triangulation is computed between the contours of the two cross sections. The three vertices of each triangle correspond to a Voronoi region and define a circle (the largest empty circle), the center of which is identified as feature point for the new intermediate cross section. (Right) A detail from the area under the chin of the cycladean idol. The newly computed feature points are positioned in 3D space and have to be projected to the corresponding 2D plane between the two existing cross sections, to form the feature poly-line of the intermediate cross section.

5.3 Representing Cross-Section Contours by G^1 Splines

The feature poly-lines we have computed for the cross sections of the point cloud provide useful information concerning the topology of the 2D point set, as they are organized point lists and may be used to reconstruct the surface of the model. But

this surface does not meet the requirements concerning the desired smoothness for the resulting model. The feature poly-lines consist of line segments and there may be parts where the gradient of neighboring line segments differs considerably. Such behavior of the method may be caused when the number of cross sections is small. If the user has chosen increased thickness for each cross section in the beginning of the processing, the parts of the point cloud will be 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 poly-lines and on the final model.

We need to represent each cross section with a smooth curve instead of a poly-line. As discussed in [163], it is a common practice to use B-Spline curves of degree 3, because they are easy to compute and capable of representing adequately most 3D objects. So we compute a closed cubic B-Spline that interpolates the points of the feature poly-line. The use of a cubic B-Spline ensures that we have G^1 continuity (instead of G^0 with the poly-line). 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 calculated according to the method described in [88] as follows.

Let the $(n+1)$ feature points of a cross section define a set of data points $Q = Q_0, Q_1, ..., Q_n$. Then a B-Spline curve of $(n+1)$ control points can be derived that passes through all the data points. The order of the B-spline curve will be $k = 4$ so that we derive a curve of degree 3. The $(n+k+1)$ knot values are defined as:

$$
\begin{aligned}
t_i &= 0 && (i = 0, 1, ..., k\text{–}1) \\
t_i &= t_{i-1} + \frac{\sum_{j=i-k}^{i-2} d_j}{\sum_{m=k}^{n+1} \sum_{j=m-k}^{m-2} d_j} && (i = k, k+1, ..., n) \\
t_i &= 1 && (i = n+1, n+2, ..., n+k)
\end{aligned}
\qquad \text{(eq 5.1)}
$$

where

$$d_j = \sqrt{|Q_{j+1} - Q_j|} \qquad \text{(eq 5.2)}$$

and Q_j are the data points already specified. The $(n+1)$ control points $P = P_0, P, ..., P_n$ have to satisfy the relation

$$Q_j = \sum_{i=0}^{n} P_i N_{i,k}(u_j) \quad (j = 0, 1, ..., n) \qquad \text{(eq 5.3)}$$

where u_j are the parameter values to be assigned to data points Q_j, and $N_{i,k}(u_j)$ are the piecewise polynomial B-spline basis functions of order k (or degree $k-1$). Any set of u_j between t_{k-1} and t_{n+1} will give a B-spline curve passing through the data points. We chose

$$u_j = \frac{t_{j+1} + t_{j+2} + ... + t_{j+k-1}}{k-1} \quad (j = 0, 1, ..., n) \qquad \text{(eq 5.4)}$$

as recommended in [88] for obtaining a smooth resulting curve. The numerical values of u_j obtained from equation eq 5.4 are substituted in equation eq 5.3, and the following simultaneous equations for P_i are derived:

$$\begin{bmatrix} \cdot & N_{i,k}(u_0) & \cdot & \cdot \\ \cdot & N_{i,k}(u_1) & \cdot & \cdot \\ \cdot & \cdots & \cdot & \cdot \\ \cdot & N_{i,k}(u_n) & \cdot & \cdot \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \cdot \\ P_n \end{bmatrix} = \begin{bmatrix} Q_0 \\ Q_1 \\ \cdot \\ Q_n \end{bmatrix} \qquad \text{(eq 5.5)}$$

We can solve for P_i from equation eq 5.5 to obtain the control points of the B-spline curve that interpolates the data points Q_i. However, there is one detail we need to consider regarding the smoothness of the curve. The process described above works for any curve, given the set of feature points for interpolation. But when it comes to constructing a closed curve, i.e. a curve where the end point coincides with the start point, it does not ensure a smooth connection at the end points. Since the boundary of the cross section forms a closed point set and the feature poly-line 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 start/end 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. Of course we are discussing about the same point, so what we need is to have the same tangent on both directions near this point, i.e. in the beginning of the curve and at the end of it.

A simple technique to achieve this is to artificially include two points in the feature poly-line, one before the start/end point and one after that, in such a position, so that we have three collinear points, with the middle point (being here the start/end point) having the same tangent in both directions.

Figure 5.5 illustrates an example curve, where the n points $P_0, P_1, P_2, ..., P_{n-2}, P_{n-1}$ are interpolated by a B-Spline. On the left, no restriction is enforced at the start/end point, so the G^1 continuity is lost. On the right, the points Q_0 and Q_1 are added in the list of feature points and the interpolating curve becomes G^1 continuous at all points, including the start/end point.

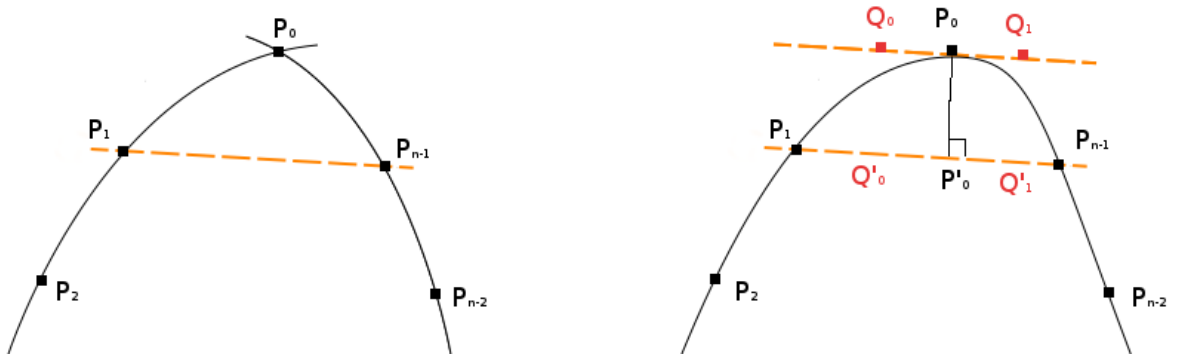


Figure 5.5: To ensure G^1 continuity on the end points we insert two more points that enforce collinearity of the three points near the start/end point of the curve.

The calculation of the points Q_0 and Q_1 depends on the position of the points P_1 and P_{n-1}. The direction of the line $P(t) = (1-t)P_1 + tP_{n-1}$ defines the direction of the line $[Q_0, P_0, Q_1]$. We chose the points Q_0, Q_1 to be the points for which the projection Q_0', Q_1' to the line $P(t)$ is in the middle of $[P_1, P_0']$ and $[P_0', P_{n-1}]$ respectively, so for the euclidean distance d we have:

$$d(Q_0', P_1) = d(Q_0', P_0') \quad and \quad d(Q_1', P_0') = d(Q_1', P_{n-1})$$

and this ensures a G^1 continuous connection at the start/end point of the closed curve. Of course, this process has to be performed before the interpolation of the B-Spline. The points Q_0 and Q_1 are considered as feature points, and the number n of points in the feature poly-line includes these two points (which, however, are not points of the initial point cloud).

After the interpolation of the B-Spline on the feature poly-line, another problem may arise, which may require the re-calculation of the continuous curve. There may be cases where some points of a feature poly-line are positioned very close to each other. When a B-spline is attempted to interpolate many data points in a limited area, it sometimes passes through the data points in an unpredictable path, such as

illustrated in figure 5.6 (Up). This effect is called **over fitting** of the curve, and a simple way to avoid it is to use fewer knots for the interpolating curve. Of course, this means that we should use fewer data points for interpolation. For this reason, our method allows the user to thin out the feature poly-line by retaining several representatives of the feature points and discard the rest of the feature points within a specific area. After thinning the feature poly-line, the interpolating B-spline should be properly computed. The user is also allowed to manually omit a specific feature point from the calculations if it would corrupt the resulting curve. Figure 5.6 (Down) shows the correctly computed part of the curve.

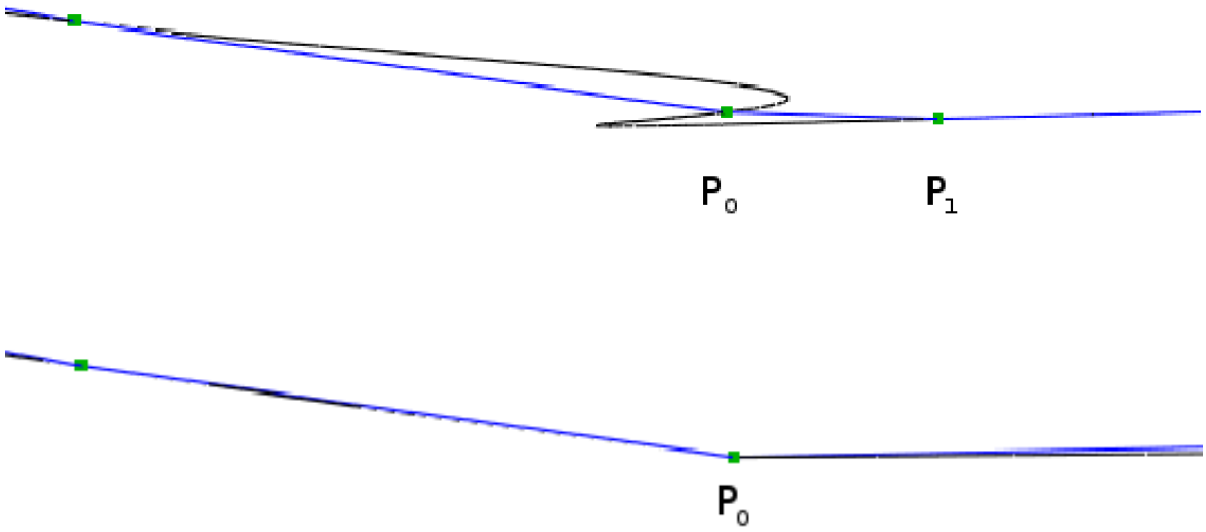


Figure 5.6: A detail of a smooth curve. (Up) If the feature points are very close to each other, the resulting curve may have irregular or even self intersecting sections due to over-fitting. (Down) Discarding selected points to thin out the feature poly-line will correct the problem of over fitting and produce accurate curves.

Once we have acquired the B-Spline 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. A sequence of the resulting curves is illustrated on figure 5.7.

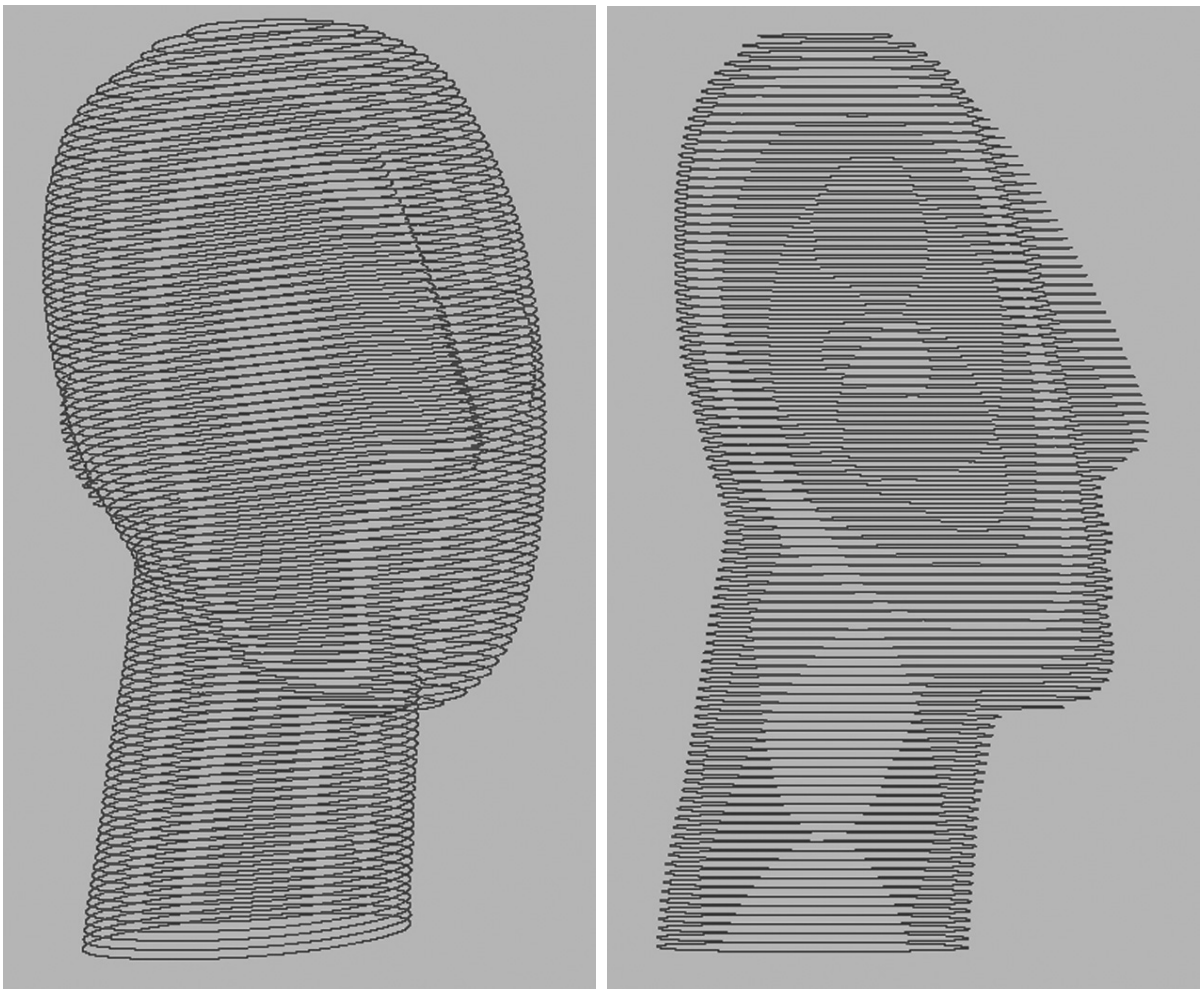


Figure 5.7: The sequence of curves of the entire cycladean idol point cloud (100 cross sections).

5.4 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 what we need is to select one point at a specific position on each (horizontal) curve (i.e. for all cross sections) and to define a new (vertical) curve

that passes through these points. That is, if we consider the cross section to be aligned horizontally, 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 curves.

A first thought would be to use the feature points, which are already available, and construct an interpolating curve that would pass through a feature point on all cross sections. This approach seems like a good idea, and it would have no additional cost concerning the computations, as the required methods are already available from the previous steps. The process would just include the construction of the vertical curves using the method described in section 5.3.

But the result of this method is not as expected, as illustrated in figure 5.8.

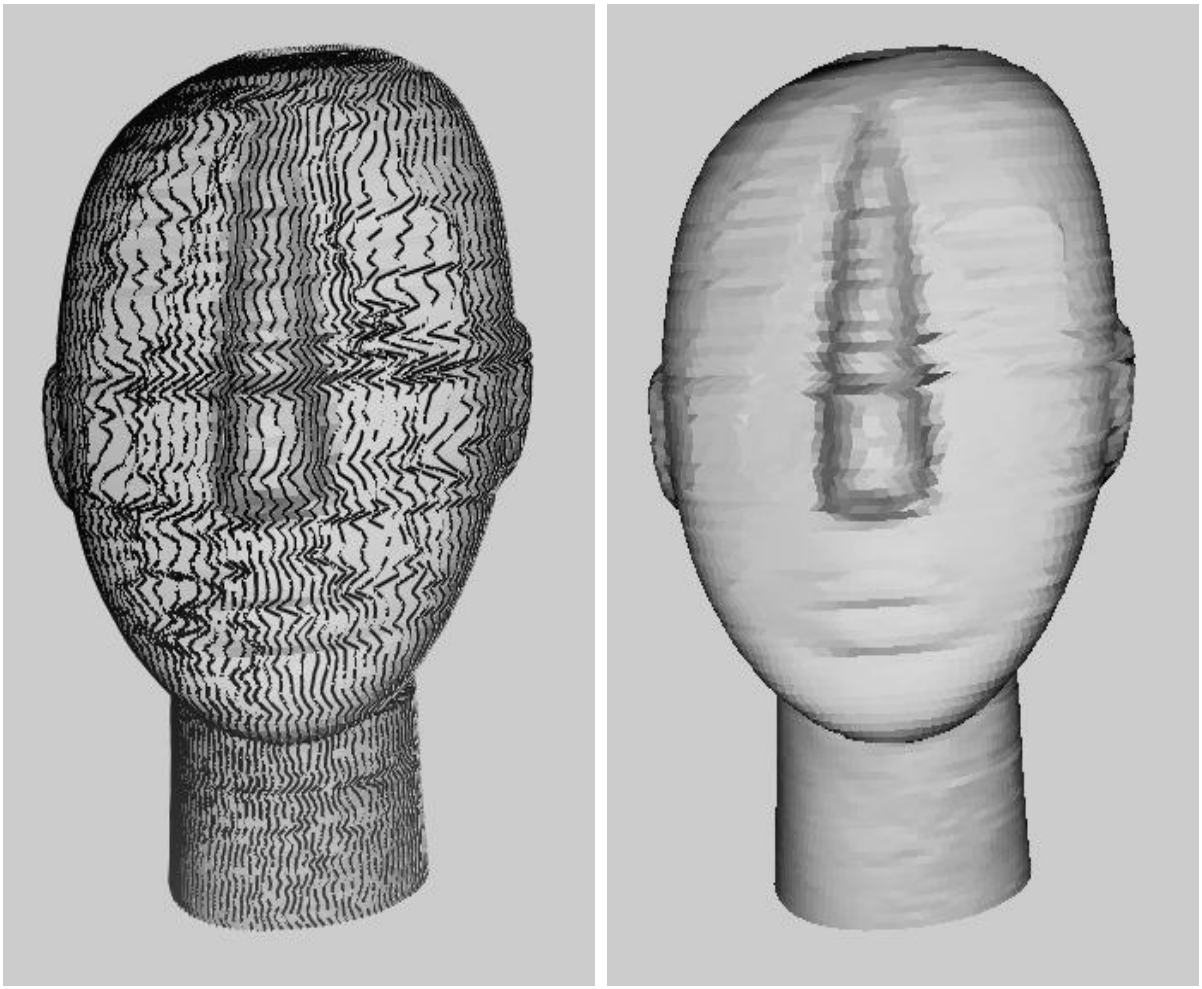


Figure 5.8: Interpolating a curve on the existing feature points does not produce good results, when the points are not properly aligned. (Left) The correspondence between the feature points. (Right) The surface of the resulting model.

The problem arises when the feature points on one cross section are not properly aligned with the corresponding feature points on the previous and next cross section. While the initial point cloud consisted of points with a uniform dispersion, the process of projecting the points on their 2D planes and the computation of the feature poly-line, have provided us with points that may be very close to each other at some regions and far from each other at other regions. Furthermore, the number of feature points may be different from one cross section to another, so there may not be a one-to-one correspondence between the feature points.

The solution to this problem is to use points other than the existing feature points of the cross sections for constructing the vertical curves and eventually to reconstruct the surface of the model. As we have already a G^1 smooth and continuous B-Spline curve available for each cross section, we may re-sample the curves and take an equal number of points with an even dispersion from each other on all cross sections.

Additionally, the start/end points of each curve also may have not been properly aligned with each other, although the initial feature points may have been ordered according to their coordinates. In the model of figure 5.9 (left), the feature points chosen for starting the feature poly-lines are the points p_i with $p_x = min(x)$ and $p_y = min(|y|)$, which means that the most left point on the $x - axis$ and the point **nearest to** $zero$ on the $y - axis$ is used as start/end point for the feature poly-line on each cross section.

By re-sampling the B-Spline curves, the start/end points are also updated, to be properly aligned to each other. Figure 5.9 (right) shows the aligned start/end points, for which we have $p_y = 0$ instead of $p_y = min(|y|)$. this means that all curves start at the most left point on the $x - axis$ and the point **exactly at** $zero$ on the $y - axis$.

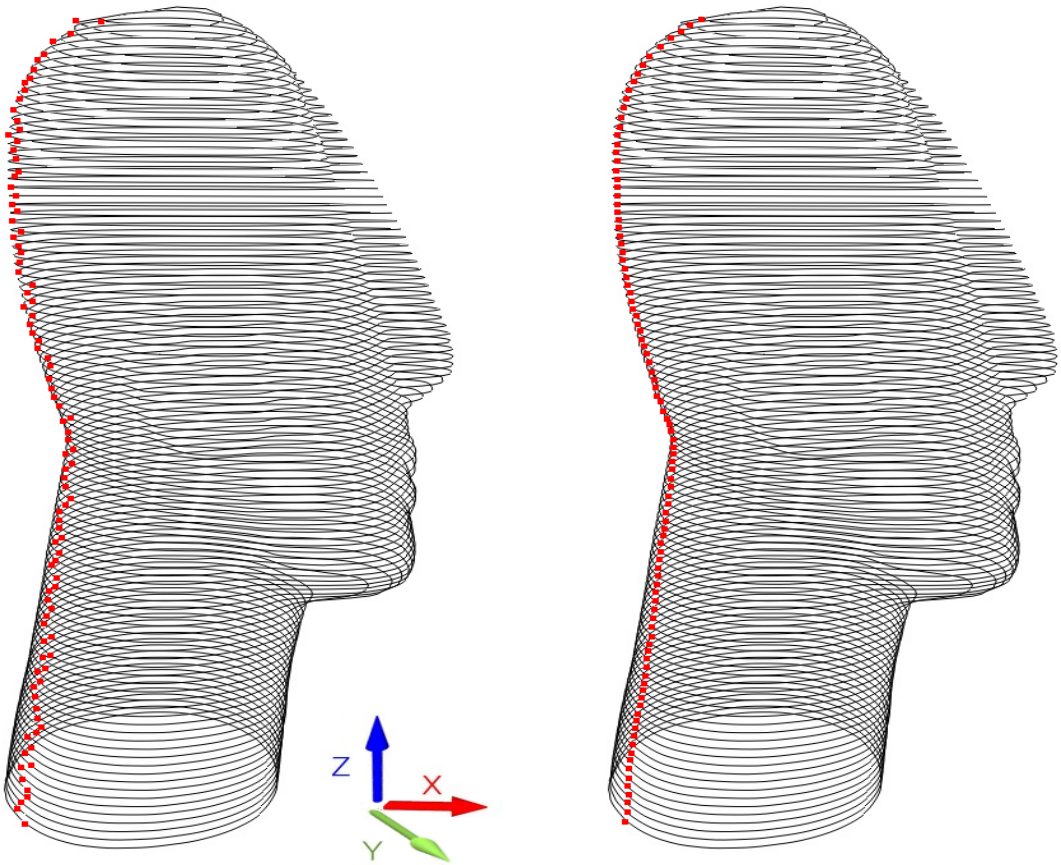


Figure 5.9: (Left) The start/end point p_i on each cross section is set to the feature point with $p_x = min(x)$ and $p_y = min(|y|)$. The points are not aligned and the surface reconstruction will suffer from irregularities. (Right) By re-sampling the curves we are able to set the start/end points p_i to the points with $p_x = min(x)$ and $p_y = 0$. The points are now aligned and the surface reconstruction becomes robust.

So far so good, but we haven't explained yet, how are the B-Splines re-sampled to acquire the aligned points?

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 [77], 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, 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.

As described in [77], for the arc length parameterization we begin with the parametric representation of a cubic spline curve

$$Q(t) = (x(t), y(t), z(t)),$$

where t is from t_0 to t_n, n is the number of spline segments, and $t_0, t_1, t_2, ..., t_n$ are the break points. The arc-length parameterization of a curve can be constructed from any other differentiable parameterization by the following two-step process:

- Compute arc length s as a function of parameter t: $s = A(t)$. Since s is a strictly increasing function of t, there is a one-to-one correspondence between s and t.
- Compute $t = A^{-1}(s)$, the inverse of the arc length function. This function is well defined and monotonically increasing for cubic splines. By substituting $t = A^{-1}(s)$ into $Q(t)$, we get a curve parameterized by arc length s, $P(s) = (x(A^{-1}(s)), y(A^{-1}(s)), z(A^{-1}(s)))$, where $s \in [0, L]$ and L is the total length of the curve.

The arc length is a geometric integration,

$$A(t) = \int_{t_0}^{t} \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2} dt, \qquad \text{(eq 5.6)}$$

where for a cubic spline,

$$\begin{cases} x(t) = a_{x,i}(t-t_i)^3 + b_{x,i}(t-t_i)^2 + c_{x,i}(t-t_i) + d_{x,i} \\ y(t) = a_{y,i}(t-t_i)^3 + b_{y,i}(t-t_i)^2 + c_{y,i}(t-t_i) + d_{y,i} \\ z(t) = a_{z,i}(t-t_i)^3 + b_{z,i}(t-t_i)^2 + c_{z,i}(t-t_i) + d_{z,i}, \end{cases}$$

where $t \in [t_i, t_{i+1}], i = 0, 1, 2, ..., n-1$, the values for x, y, and z are of class C^2 on $[0, L]$. In general, the integral of equation eq 5.6 cannot be computed analytically. Therefore, the arc-length parameterization for cubic spline curves cannot be expressed as a combination of elementary functions and must be evaluated numerically.

This method computes the approximation curve in three steps. First, the arc lengths of all the cubic segments in the input spline curve, $Q(t)$, are computed and summed to determine the arc length L of $Q(t)$. The second step is to find $m+1$ points equally spaced along $Q(t)$. The third step is to compute a new spline curve using the equally spaced points as knots.

The result is an approximately arc-length parameterized piecewise spline curve divided into m cubic segments. The arc length of each spline segment on the input curve is

$$l_i = \int_{t_i}^{t_{i+1}} \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2} dt,$$

where i varies from 0 to $n-1$ and n is the number of spline segments in the original curve. Thus, the arc length of the whole curve is $L = \sum_{i=0}^{n-1} l_i$.

Using the bisection method, we compute $m+1$ equally spaced points on $Q(t)$ located at distances $0, \tilde{l}, 2 \cdot \tilde{l}, \cdots, m \cdot \tilde{l}$ from the start of the curve, where $\tilde{l} = L/m$ is the length of each segment in the output curve. These points are defined by the parameter values $\tilde{t}_0, \tilde{t}_1, ..., \tilde{t}_m$, which satisfy the following integration,

$$\int_{t_0}^{\tilde{t}_i} \frac{ds}{dt} dt = i \cdot \tilde{l}, \qquad \text{(eq 5.7)}$$

where $i = 0, 1, ..., m$, s is arc length, and t is the parameter of the spline functions.

The value of $\tilde{t}_i$ can be computed in two steps. The first step is to find a spline segment indexed by j which satisfies $\sum_{p=0}^{j-1} l_p \leq i \cdot \tilde{l} < \sum_{p=0}^{j} l_p$. This condition ensures that $t_j \leq \tilde{t}_i < t_{j+1}$.

Equation eq 5.7 is written as,

$$\int_{t_0}^{\tilde{t}_i} \frac{ds}{dt}dt = \int_{t_0}^{t_j} \frac{ds}{dt}dt + \int_{t_j}^{\tilde{t}_i} \frac{ds}{dt}dt = \sum_{p=0}^{j-1} l_p + \int_{t_j}^{\tilde{t}_i} \frac{ds}{dt}dt = i \cdot \tilde{l}$$

The second step is to compute t_i such that

$$\int_{t_j}^{\tilde{t}_i} \frac{ds}{dt}dt = i \cdot \tilde{l} - \sum_{p=0}^{j-1} l_p$$

where t_i is on the cubic spline segment starting with parameter value t_j. The second step is accomplished with the bisection method. We suppose $t_{left} = t_j$ and $t_{right} = t_{j+1}$. The interval $[t_{left}, t_{right}]$ contains the solution $\tilde{t}_i$. This interval is bisected into two subintervals $[t_{left}, t_{middle}]$ and $[t_{middle}, t_{right}]$, where $t_{middle} = (t_{left} + t_{right})/2$. We can calculate the arc length between $[t_j, t_{middle}]$ as $\Delta s = \int_{t_j}^{t_{middle}} \frac{ds}{dt}dt$. The solution lies in the upper subinterval $[t_{middle}, t_{right}]$ if $\Delta s < i \cdot \tilde{l} - \sum_{p=0}^{j-1} l_p$. Otherwise, the solution lies in the lower subinterval $[t_{left}, t_{middle}]$. This bisection process is repeated until a required error tolerance for arc length is achieved.

With the above bisection method, we get $\tilde{t}_0, \tilde{t}_1, \cdots, \tilde{t}_m$ that divide the original curve into equal arc-length segments. Using the original cubic spline function, we then compute the evenly spaced points $(\tilde{x}_0, \tilde{y}_0, \tilde{z}_0), (\tilde{x}_1, \tilde{y}_1, \tilde{z}_1), \cdots, (\tilde{x}_m, \tilde{y}_m, \tilde{z}_m)$ at arc-lengths $s_0 = 0, s_1 = \tilde{l}, s_2 = 2 \cdot \tilde{l}, \cdots, s_m = m \cdot \tilde{l}$. We re-parameterize the spline curve by interpolating $[(s_0, \tilde{x}_0), (s_1, \tilde{x}_1), \cdots, (s_m, \tilde{x}_m)]$, $[(s_0, \tilde{y}_0), \cdots, (s_m, \tilde{y}_m)]$ and $[(s_0, \tilde{z}_0), \cdots, (s_m, \tilde{z}_m)]$. In this interpolation, we interpolate x, y and z to arc length s and get the cubic spline functions in equation eq 5.8. Our goal is to have the following result for $s \in [0, L]$:

$$\sqrt{(\tilde{x}'(s))^2 + (\tilde{y}'(s))^2 + (\tilde{z}'(s))^2} = 1.0$$

Therefore, the magnitude of the beginning tangent vector and the magnitude of the ending tangent vector should be 1.0. The new curve is

$$\begin{cases} \tilde{x}(s) = \tilde{a}_{x,i}(s - s_i)^3 + \tilde{b}_{x,i}(s - s_i)^2 + \tilde{c}_{x,i}(s - s_i) + \tilde{d}_{x,i} \\ \tilde{y}(s) = \tilde{a}_{y,i}(s - s_i)^3 + \tilde{b}_{y,i}(s - s_i)^2 + \tilde{c}_{y,i}(s - s_i) + \tilde{d}_{y,i} \\ \tilde{z}(s) = \tilde{a}_{z,i}(s - s_i)^3 + \tilde{b}_{z,i}(s - s_i)^2 + \tilde{c}_{z,i}(s - s_i) + \tilde{d}_{z,i}, \end{cases} \quad \text{(eq 5.8)}$$

where $s \in [s_i, s_{i+1}], i = 0, 1, 2, ..., m - 1$, and the values for $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ are of class C^2 on $[0, L]$. The tangent vectors of the derived curve at the beginning point and the ending point are set to be equal to the normalized tangent vectors of the original curve at the beginning point and the ending point, respectively.

This method provides an arc-length parameterized spline curve with m equal-length spline segments. When we apply it to the B-Splines of all cross sections on our model, we get a new set of sample points, which are evenly dispersed on the curves according to their arc-length.

Unlike the contour triangulation method described in section 5.2, the new sample points do not have to be combined across the cross sections, to find a proper correspondence between the points according to shortest distance. In this case the arc-length method has provided an ordered point list on each cross section, with equal number of points for all cross sections, which are positioned at equal arc-length intervals that ensure a one-to-one correspondence without any more computations. This means that the new points are aligned to each other across the cross sections, and we have the same number of points on all cross sections.

Furthermore, we know that for the n sample points we have acquired on each cross section, we can construct n vertical curves that use the same point on each cross section. That is, the vertical curve v_i will pass through point p_i on all cross sections. The vertical curves are also B-Spline curves, which are acquired with the use of the interpolation method described in section 5.3. Unlike the B-Splines that represent the cross sections, these curves are not closed on their end points, and no restriction is required concerning the continuity and smoothness in the beginning and at the end of the curves. Figure 5.10 shows a detail of a model where the cross sections P and Q have been re-sampled and the vertical curves have been constructed. The vertical curve on the left passes through all cross sections on point $0(p_0, q_0, ...)$, the vertical curve on the middle passes through all cross sections on point $1(p_1, q_1, ...)$, the vertical curve on the right passes through all cross sections on point $2(p_2, q_2, ...)$, and so on.

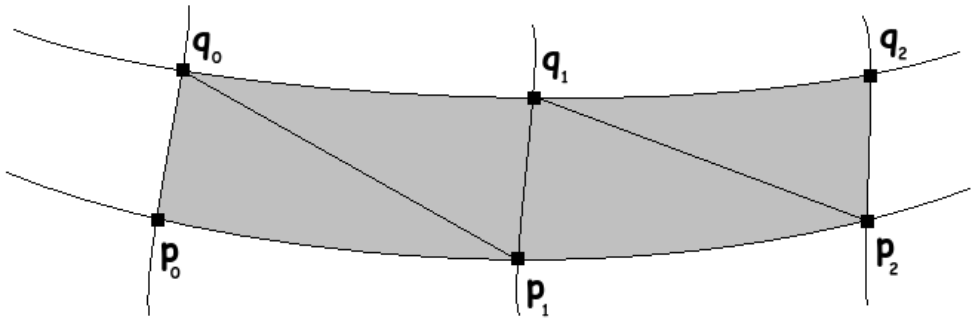


Figure 5.10: Interpolating vertical B-Splines on the newly re-sampled feature points according to the arc-length method ensures a proper representation of the model surface with a one-to-one correspondence between the points across all cross sections.

For simplicity of the computations, since the arc length parameterization uses the bisection method on the curve segments, we have chosen the number of the sample points to be 2^n (n the number of iterations), which means that the arc-length method will make n bisections on the initial curve. In other words, the number of vertical curves we request, which also represents the level of detail on the surface reconstruction, will be $2, 4, 8, 16, 32, 64, 128, ...$ and so on.

At the end, for representing the surface of the model as a simple triangle mesh, we may use the four neighboring points defined by two adjacent cross sections and two adjacent vertical curves, to form a pair of triangles (or one quad) for rendering purposes, as illustrated in figure 5.10.

Figure 5.11 summarizes the final steps of the reconstruction method. Figure 5.11 (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. Figure 5.11 (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. Figure 5.11 (c) shows the B-Splines and the vertical curves combined together. For visualization purposes only a 20% of the actual curves are rendered. Figure 5.11 (d) shows the reconstructed surface of the model.

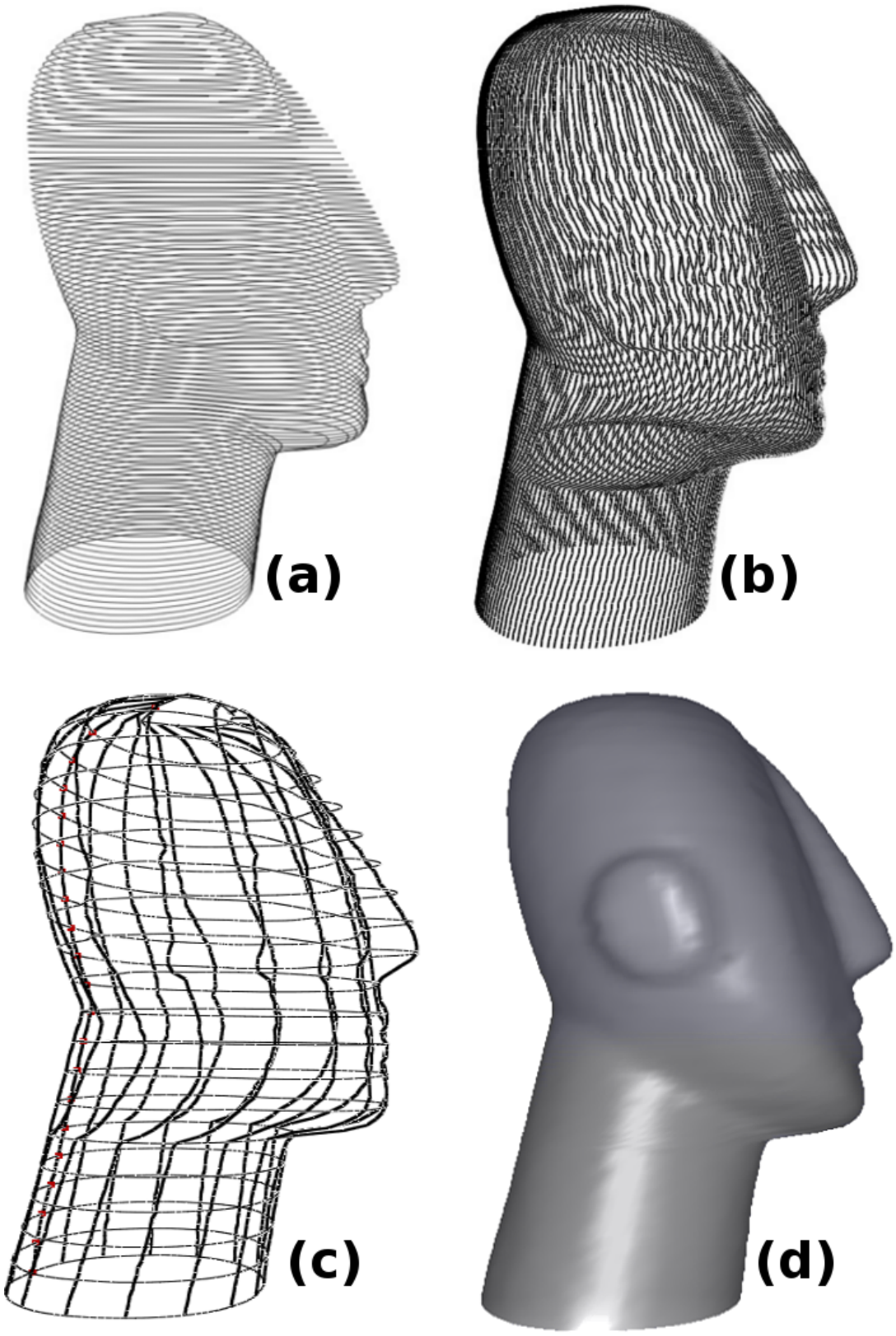


Figure 5.11: The resulting model: (a) the B-Spline curves of the cross sections, (b) the vertical curves, (c) a thinned visualization of the cross sections combined with the vertical curves, and (d) the reconstructed surface.

Chapter 6

Editing Free Form Models

6.1 Introduction

6.2 Applying Free Form Transformations

6.3 Implementation and Examples

6.1 Introduction

We now have an editable representation of the point cloud at our disposal and we may perform high level modifications on the model. The nature of the model itself is crucial for the computations we are about to apply. Mechanical parts will be treated with a different approach than free form objects.

When dealing with mechanical parts, editing is usually constrained to the features of the object. Such point clouds usually represent objects with geometric primitives (e.g. spherical or cylindrical holes or extrusions). In this case the modifications could be directed according to the properties of these primitives (e.g. modify radius, height or width of a feature). For example, in a part with a cylindrical shape it would be reasonable to modify the radius or the height of the cylinder.

On the other hand, when dealing with free-form objects, 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. The user may perform arbitrary modifications to the model, limited only by their imagination

and of course the correctness of the resulting model. In this case, however, the editing procedure can become difficult due to complex, highly variable morphologies.

6.2 Applying Free Form Transformations

In our implementation, we deal with free form transformations, which are applied to a specified number of selected cross sections. The user has the option 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. That is, all points that are associated with a given cross section, including cloud points, feature points, control points of the interpolating B-Spline, Voronoi vertices etc. This 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 one or more cross sections at once. As we mentioned in section 3.2.4, many cross sections may be selected by the user. When applying modifications, we may apply a given transformation either on the current cross section (i.e. the cross section we are currently working on), or all the selected cross sections. The user may select as many cross sections as they desire, 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. We will then 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 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).

We have included another option, which 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. With this option enabled, given a list of selected cross sections, the transformation will be partially applied to the cross sections, depending

on their index in the list of 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 evaluate a mapping function that provides a function of the proportional transformation to each cross section according to its index in the list of cross sections. This mapping function $f(h)$ is evaluated for each selected cross section separately. The index i of the cross section in the list of n selected cross sections defines the parameter of the function, in the form of $h = (i-1)/(n-1)$, where i is the current cross section and n is the number of selected cross sections. The value of h lies in the interval $[0..1]$, so for the first cross section we have $i = 1 \;\rightarrow\; h = 0$. For the second cross section we have $i = 2 \;\rightarrow\; h = 1/(n-1)$, and so on, until $i = n-1 \;\rightarrow\; h = (n-2)/(n-1)$ for the cross section before the last, and $i = n \;\rightarrow\; h = 1$, which corresponds to the last selected cross section.

As an example, we may set a mapping function $f(h) = \sin(h\pi)$, to apply a transformation using the $sine$ function and produce some kind of a wave effect on the cross sections. 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 meets their requirements.

6.3 Implementation and Examples

Our method has been implemented and tested under the Microsoft Visual C++ programming environment [164] using the Qt UI framework [165]. The computational geometry calculations were performed using the Qhull library [157]. Evaluations of the mapping function were made with the simple arithmetic expression evaluator developed by Robert B. Stout [166].

Figure 6.1 illustrates parts of the UI that have been implemented for testing our

method. Some examples of complex transformations applied to the entire point cloud of the Cycladic idol are illustrated in figure 6.2. Figure 6.3 shows transformations applied to certain parts of the same model. Figure 6.5 shows a realistic example of transformations applied to the drill bit model, which result in some variants of the original object that could potentially be applied in practical applications.

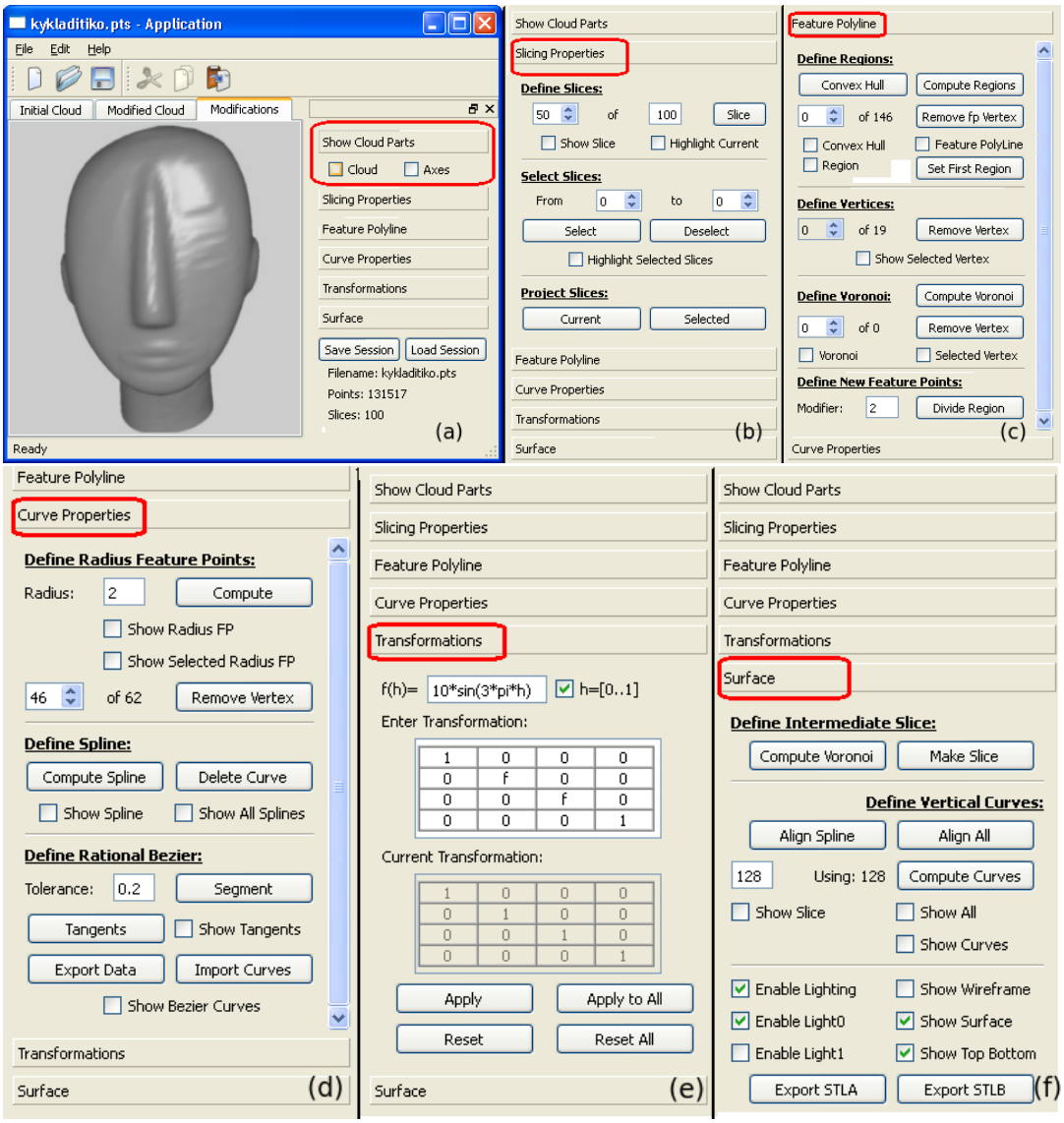


Figure 6.1: The User Interface of our implementation. From left to right: (a) The entire window, with save-load options, (b) Cross section operations, (c) Feature polyline properties, (d) B-spline curve operations, (e) Transformation properties, and (f) Surface reconstruction properties.

The following examples show how the free form transformations are applied to the models of the cycladean idol and the twist drill bit. In the examples that represent

variations of the cycladean idol, the point cloud was sliced into 100 cross sections, numbered from 0 to 99 (cross section 0 at the bottom, cross section 99 at the top). In the example of the twist drill bit, the point cloud was sliced into 200 cross sections, numbered from 0 to 199 (cross section 0 at the bottom, cross section 199 at the top).

The mapping function $f(h)$ is applied to each of the selected cross sections according to their index in the list of selected cross sections. For example, in the transformations of figure 6.1 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 to each cross section by multiplying the elements of the cross section with the appropriate transformation matrix.

In the following examples we have used the transformation matrices S_{xy}, S_z and T_x that scale and translate the cross sections in the corresponding directions:

$$S_{xy} = \begin{bmatrix} f(h) & 0 & 0 & 0 \\ 0 & f(h) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad S_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f(h) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad T_x = \begin{bmatrix} 1 & 0 & 0 & f(h) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Of course, any other transformation matrix may be used for modifying the model. The function $f(h)$ doesn't even have to be the same at all matrix elements. We could e.g. set a scaling matrix S'_{xy} that performs scaling on the $x - axis$ with a different factor than the $y - axis$. Furthermore, a complex mathematical expression could be evaluated for each element, similar to the evaluation we performed in $f(h)$.

In the examples of the following transformation matrices, S'_{xy} performs scaling on x with a factor $f(h)$ and scaling on y with a factor $2 * f(h)$, while ST_{xy} performs scaling on x with a factor $f(h)$ and translation on y with a factor $sin(f(h))$:

$$S'_{xy} = \begin{bmatrix} f(h) & 0 & 0 & 0 \\ 0 & 2 * f(h) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad ST_{xy} = \begin{bmatrix} f(h) & 0 & 0 & 0 \\ 0 & 1 & 0 & sin(f(h)\pi) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Any combination is possible and will produce a variation of the model. However, the resulting model may or may not make sense, as the transformation matrix may perform modifications that could cause the model to deform in unnatural manner.

Whether the model makes sense or not, it is up to the proper definition of the transformation matrix. For this reason, it is essential for the user to have some knowledge on homogeneous transformations.

Example 1: Transformations applied to the entire point cloud of the cycladean idol.

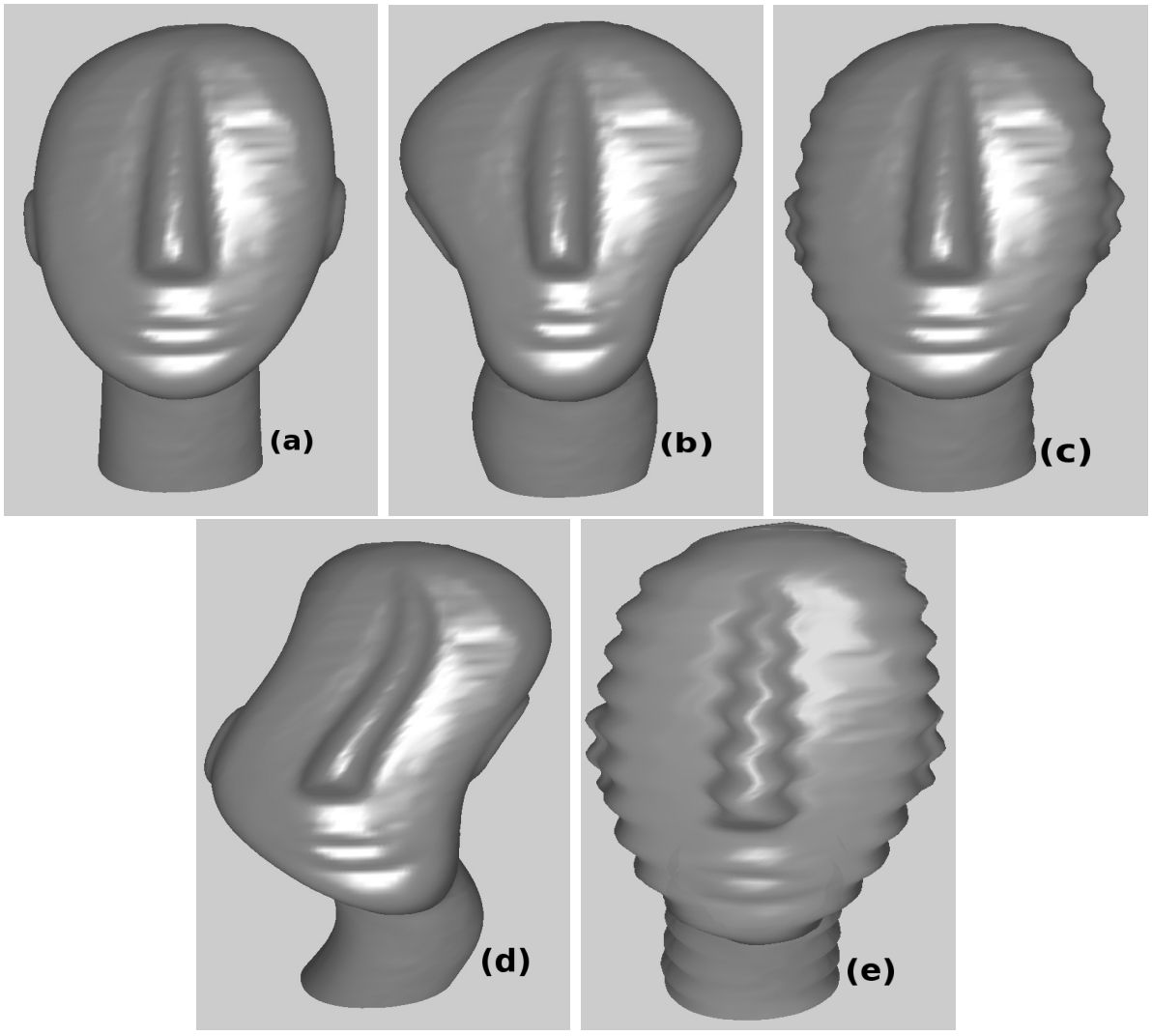


Figure 6.2: Transformations applied to the entire object, (a) plain object, without any transformation, (b) large wave scaling, (c) small wave scaling, (d) large wave translation, and (e) small wave translation.

The following table describes the mapping function $f(h)$ used for the transformations applied to the models of figure 6.2.

Figure	Selected Cross Sections	Mapping Function	Applied Transformation
Fig. 6.2 (a)	-	-	None (Original Model)
Fig. 6.2 (b)	[0..99]	$f(h) = \frac{5+\sin(10h)}{6}$	Scale S_{xy}
Fig. 6.2 (c)	[0..99]	$f(h) = \frac{50+\sin(100h)}{51}$	Scale S_{xy}
Fig. 6.2 (d)	[0..99]	$f(h) = 10\sin(3h\pi)$	Translate T_x
Fig. 6.2 (e)	[0..99]	$f(h) = 1.5\sin(30h\pi)$	Translate T_x

Table 6.1: Details for transformations on Fig. 6.2.

Example 2: Transformations applied to selected cross sections of the cycladean idol.

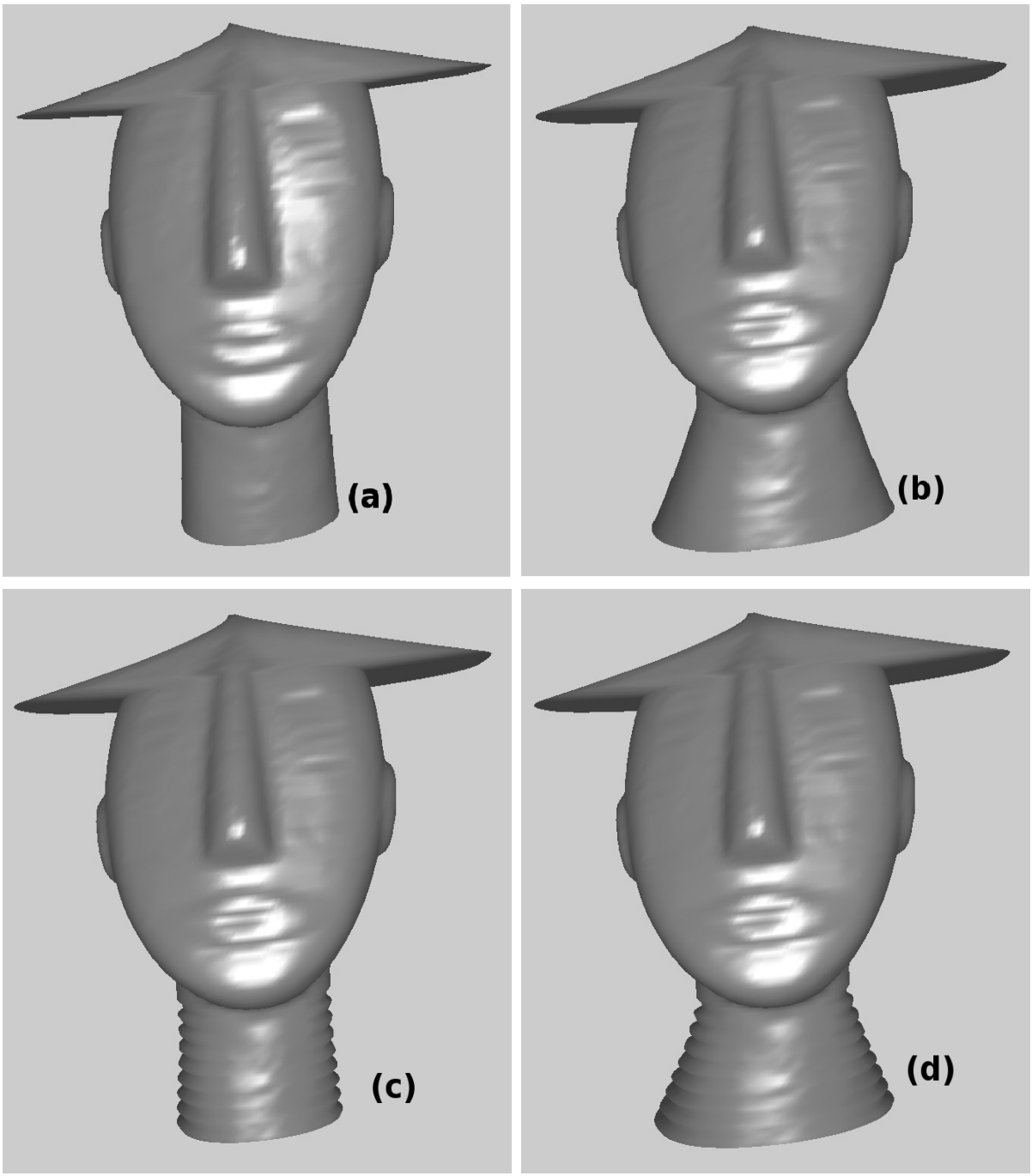


Figure 6.3: Transformations applied to parts of the object, (a) 'Chinese hat', (b) 'Chinese hat' + 'growing neck', (c) 'Chinese hat' + 'wavy neck', and (d) 'Chinese hat' + 'growing neck' + 'wavy neck'.

The next table describes the mapping function $f(h)$ used for the transformations applied to the models of figure 6.3.

Figure	Selected Cross Sections	Mapping Function	Applied Transformation
Fig. 6.3 (a)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
Fig. 6.3 (b)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
	[0..25]	$f(h) = 1,5 - 0.5h$	Scale S_{xy}
Fig. 6.3 (c)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
	[0..25]	$f(h) = \frac{20+\sin(50h)}{21}$	Scale S_{xy}
Fig. 6.3 (d)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
	[0..25]	$f(h) = 1,5 - 0.5h$	Scale S_{xy}
	[0..25]	$f(h) = \frac{20+\sin(50h)}{21}$	Scale S_{xy}

Table 6.2: Details for transformations on Fig. 6.3.

Example 2 (cont.): Transformations applied to selected cross sections of the cycladean idol.

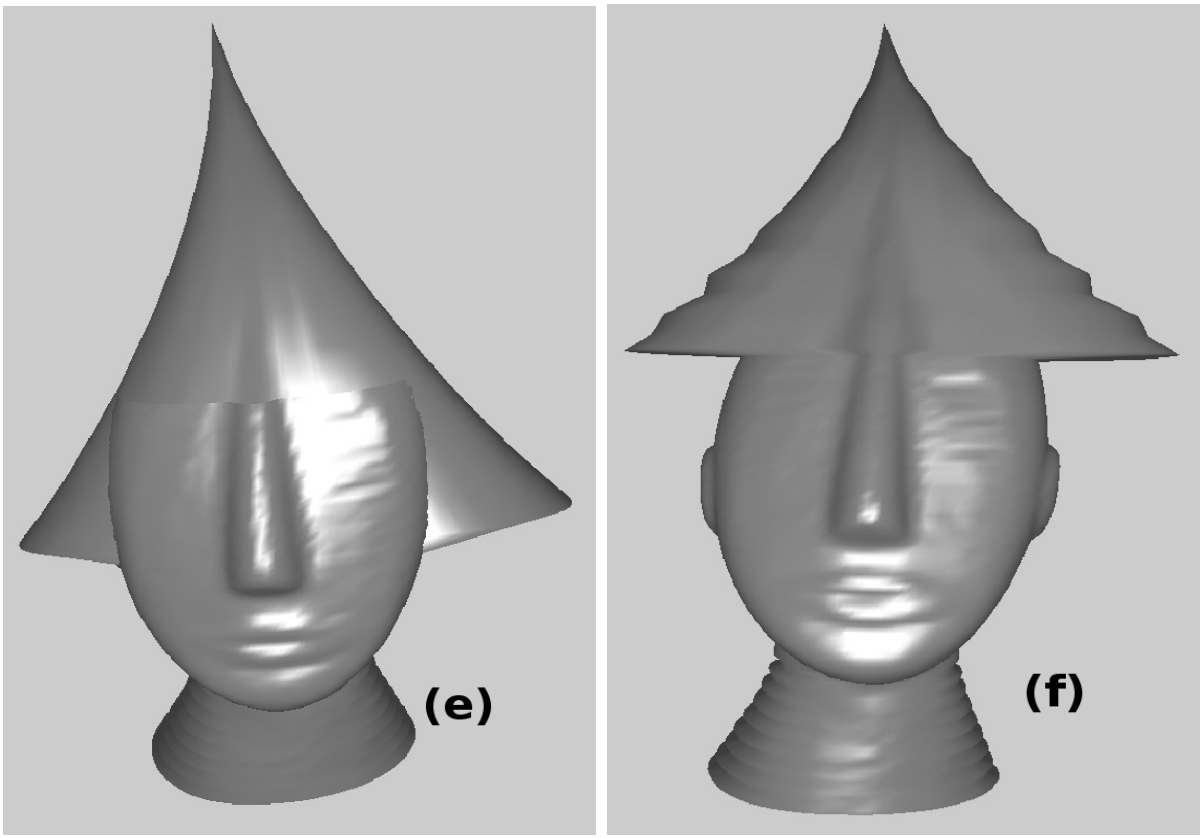


Figure 6.4: Transformations applied to parts of the object, (e) 'Asian Monk hat', (f) 'Gandalf the grey'.

The mapping function $f(h)$ used for the transformations applied to the models of figure 6.4.

Figure	Selected Cross Sections	Mapping Function	Applied Transformation
Fig. 6.4 (e)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
	[85..99]	$f(h) = 2h$	Scale S_z
	[0..25]	$f(h) = 1,5 - 0.5h$	Scale S_z
	[0..25]	$f(h) = \frac{20+\sin(50h)}{21}$	Scale S_{xy}
Fig. 6.4 (f)	[85..99]	$f(h) = 2 - 2h$	Scale S_{xy}
	[85..99]	$f(h) = 1 + h$	Scale S_z
	[85..99]	$f(h) = \frac{20+\sin(50h)}{21}$	Scale S_{xy}
	[0..25]	$f(h) = 1,5 - 0.5h$	Scale S_{xy}
	[0..25]	$f(h) = \frac{20+\sin(50h)}{21}$	Scale S_{xy}

Table 6.3: Details for transformations on Fig. 6.4.

Example 3: Transformations applied to selected cross sections of the twist drill bit.

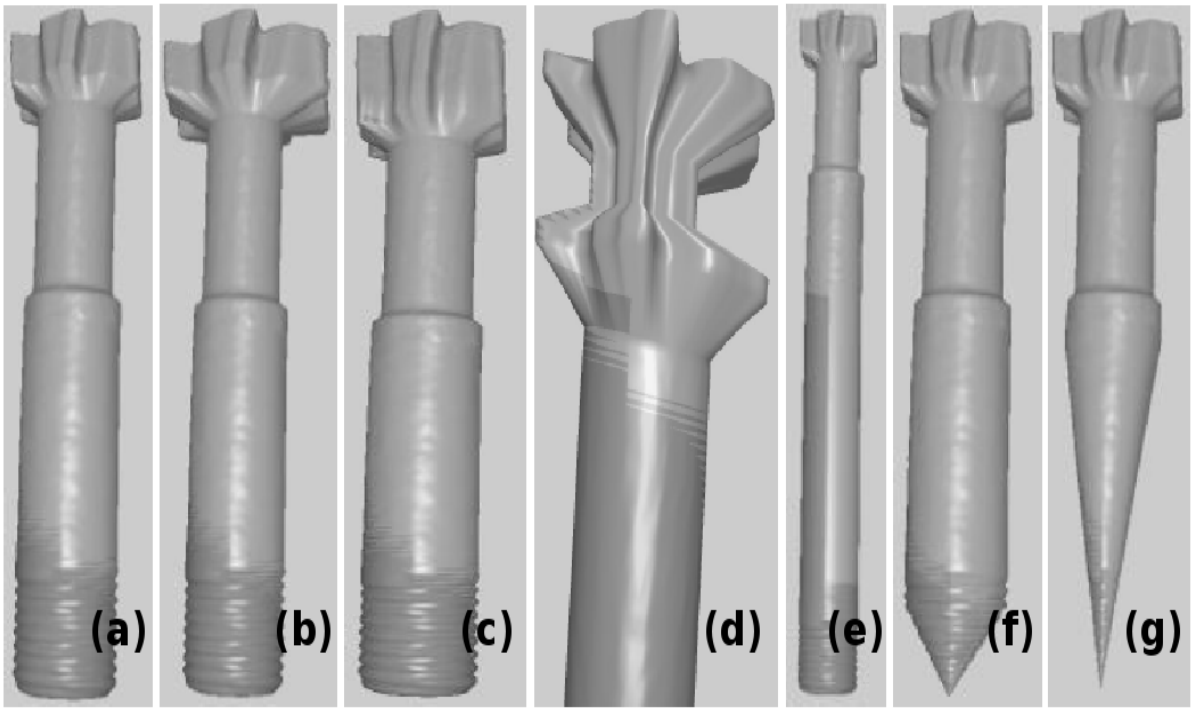


Figure 6.5: Transformations applied on the twist drill bit point cloud.

The following table describes the mapping function $f(h)$ used for the transformations applied to the models of figure 6.5. Note that when $f(h)$ is a constant function, the implied transformation is applied to all selected cross sections evenly.

Figure	Selected Cross Sections	Mapping Function	Applied Transformation
Fig. 6.5 (a)	-	-	None (Original Model)
Fig. 6.5 (b)	[175..199]	$f(h) = 1.5$	Scale S_{xy}
Fig. 6.5 (c)	[175..199]	$f(h) = 1.5$	Scale S_z
Fig. 6.5 (d)	[178..184]	$f(h) = 1 - 0.5h$	Scale S_{xy}
	[185..190]	$f(h) = 1.5$	Scale S_{xy}
	[191..196]	$f(h) = 0.5 - 0.5h$	Scale S_{xy}
Fig. 6.5 (e)	[0..50]	$f(h) = -50$	Translate S_z
Fig. 6.5 (f)	[0..25]	$f(h) = h$	Scale S_{xy}
Fig. 6.5 (g)	[0..100]	$f(h) = h$	Scale S_{xy}

Table 6.4: Details for transformations on Fig. 6.5.

All the examples presented above use the transformation matrices S_{xy}, S_z, T_z mentioned earlier. However, the use of any other transformation is allowed, as the transformation matrix is a user defined parameter which can be set arbitrarily. It is up to the creativity of the user to provide a proper transformation matrix that will produce impressive yet reasonable models.

Chapter 7

Editing Operators for Cross-Sectional Data-Sets

7.1 Introduction

7.2 Properties of the Input Model

7.3 Editing Operators

7.3.1 The Curve of Centroids

7.3.2 Alignment Operator

7.3.3 Displacement Operator

7.3.4 Elastic Operator

7.3.5 Inflation Operator

7.3.6 Placing the curve of centroids on a parametric function

7.3.7 Cross-Sectional Free-Form Editing

7.3.8 Limitations

7.4 Implementation and Examples

7.1 Introduction

In this Chapter, we report on the development of a cross-sectional 3D model editor. The models we work with have properties often encountered in medical data-sets (e.g. CT scans of arteries or internal organs). However, our tool-set can be applied to

both engineered objects (e.g. mechanical parts) or free-form objects (e.g. human or animal figures) for various purposes.

When dealing with mechanical parts, editing is usually restricted to adjusting the features of the object. For example, in a part with a cylindrical shape it is often required to modify the radius or the height of the cylinder. In such cases the modification of the model is facilitated by the object features.

When dealing with free-form objects, the user may perform arbitrary modifications to the model, limited only by the creativity of the user and the robustness of the resulting model. In this case, however, the editing process can become difficult due to complex, highly variable topology. Vascular anatomies, for example, are patient-specific and cannot be easily built as combinations of a given number of mathematical primitives or created by a sequence of manufacturing operations [167].

7.2 Properties of the Input Model

The input for our tool suite should be generic enough, so that it would be easily used and to be applied to several applications. As we discussed in chapter 3, the simplest unstructured form of data-set is the point cloud of its surface scan. The only information a point cloud carries is the $[x, y, z]$ coordinates of a set of points that lie on the boundary of the model. Depending on the acquisition method used and the density of the scan, a point cloud may describe the topology of the boundary accurately at all parts of the model (e.g. 3D laser scanners) [9, 8], or describe only feature points of the object that lie on specific parts of the model (e.g. medical CT scans) [150, 21]. The case of medical CT scans is considered sensitive, as the direction and the density of of the scan are predefined, and cannot be refined after the data-set has been acquired.

In chapters 3, 4 and 5, we described the details on how to extract an editable CAD model from an unstructured point cloud. To briefly recap, we divide the 3D point cloud in cross-sectional slices, and treat each slice as an individual 2D point set. Using properties of the convex hull and the Voronoi diagram of the slice points, we define a set of representative feature points that describe the boundary of the slice points accurately. These feature points form a closed curve of continuity G^0, which is called a feature poly-line. Subsequently, a G^1 B-Spline curve is constructed,

which interpolates the poly-line and provides a smooth boundary representation. The contours of all cross sections are combined, and a number of points (depending on the required level of detail) are selected on each B-Spline, which are subsequently used for creating a mesh that represents the reconstructed surface with an editable model.

If the input point cloud was acquired from a medical CT scan, like the example shown in figure 7.1, it will probably be already sliced into cross sections, and the slice points will already form a feature poly-line. In this case we can omit all computations up to this point and define G^1 B-Splines from these point-sets.

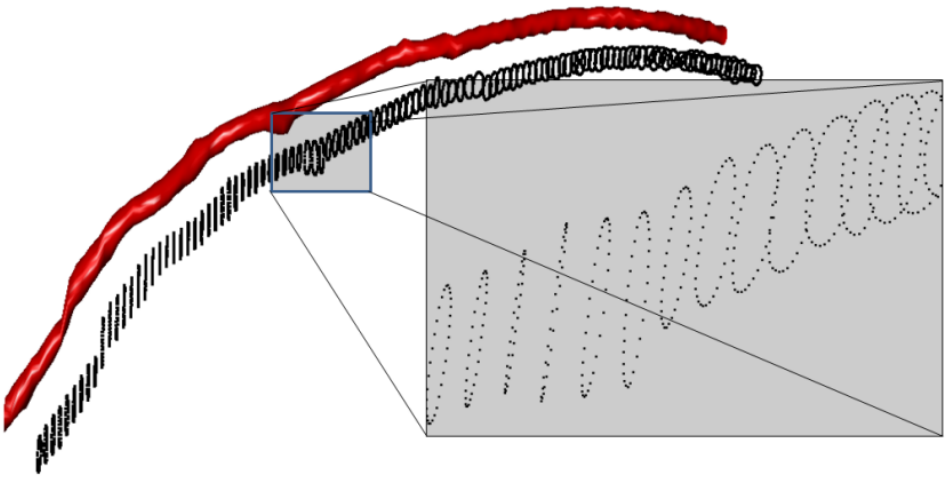


Figure 7.1: The model of an artery, and the points that came as input. **Input**: A 3D point set of an artery, organized in 2D slices of 80 points, provided from a CT Scan. **Output**: A collection of closed NURBS curves that interpolate the points of each slice accurately. These curves are used for reconstructing the surface of the model (red surface on the top left).

On the other hand, there might be some cases where the existing slices do not provide the proper information for the model. In such cases, we have the option to treat the model as an unstructured point cloud, and redefine the slices entirely. For example, we can use a different slicing direction, to align the model properly according to a specified axis, or even use variable thickness on the slices, to provide higher level of detail to some parts of the model than the rest. The model itself, however, may have some irregularities, as illustrated in the top right region of the model in figure 7.2 (back). These irregularities are not the result of defective processing, but they correspond to irregularities present on the real tissue. We need to retain these

features on the reconstructed model, as they may be essential to the final model.

7.3 Editing Operators

Editing 3D models has been addressed in several ways, following several different approaches. Editing based on features, cross-sections and geometric constraints are common in traditional CAD systems such as AutoCAD and ProEngineer. They provide robustness and accuracy but are often difficult to solve and require advanced knowledge of geometry and feature-based editing.

Free form editing usually employs mesh processing, morphing, deformations, free-form editing techniques. It also includes the trivial case of modifying individual points, but this type of editing is considered low level editing and does not provide the user with an adequate editing tool-set.

Our method offers high level free-form editing tools, and also a set of operators that perform deformations on the model. To facilitate editing we derive a skeleton of the model by means of a NURBS curve that interpolates all slice centroids. This enables the application for local or global transformations using this skeleton as a reference. These transformations may be applied as a whole, or change as we move along the skeleton. We call this skeleton the curve of centroids.

7.3.1 The Curve of Centroids

The curve of centroids is defined as a cubic B-Spline curve that interpolates the centroids of each cross section. To derive this curve, we compute the centroid of each cross section, i.e. the mean point of the slice points for each slice. We compute all the centroids and then we compute a cubic B-Spline which interpolates these centroids. We use a cubic B-Spline curve because it is easy to compute and capable of representing adequately most 3D shapes. This curve is shown in Figure 7.2 (Front).

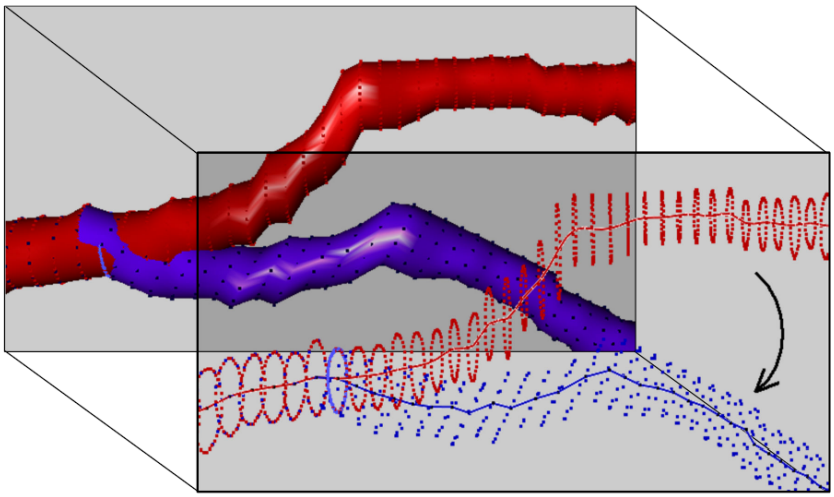


Figure 7.2: (Front) The curve that interpolates the centroids of the slices can be edited to give the model new features (Red: initial curve - Blue: edited curve). (Back) The surface of the model is adjusted according to the modification of the curve of centroids.

The curve of centroids as a concept is similar to the centerline described in [168, 169, 170, 85]. However, the two representations should not be considered as the same feature, as the point sets used for the computation of each curve are different: The centerline is extracted directly from all points of the CT scan, while our curve of centroids is computed from the final filtered point cloud of the model. So, the curve of centroids is indirectly evaluated from the CT scan, which means that it might have some differences from the centerline.

The newly acquired curve of centroids can be used to apply modifications to the model. Instead of applying transformations on the surface of the model, we may easily deform the curve of centroids. Then, the surface of the model, which is derived from the slice points, will follow the modification of the curve of centroids, as the slice points will constantly maintain their relative positioning around their centroid. This means that if e.g. a centroid is translated to another position, the corresponding slice points will follow the same transformation, to maintain their relative position to their centroid.

The motivation for implementing such an operator, i.e. one that is applied to the curve of centroids, was the fact that for medical data sets such as arteries and veins, the skeleton is an essential feature, which could be used for editing the model. Several medical applications, such as bypass surgery or stent insertion, make use of

this feature and depend on its behavior. Since the curve of centroids defines a skeleton of such models, it can be used for redefining and modifying the path of the skeleton and the shape of the model.

Various transformations can be applied to the curve of centroids. For example, parts of the curve may be displaced with a transformation that would translate the selected centroids to a new position. Or a part of the curve may be scaled according to a given point of reference. Parts of the curve could also be rotated around a specified axis. The slice points will always preserve their relative positioning around their corresponding centroids, and the surface of the model would follow the same transformation as the curve of centroids.

The curve of centroids is a high level feature we have extracted from the unstructured point cloud. However, this feature cannot be used by itself to perform editing on the model, as the applicability of the modifications is still limited and their complexity makes them unfit for CAD end-users. A set of high level parameters is required to describe specific properties of the model, which will be used for the editing operations. The operators we describe in the next sections make use of such parameters to perform modifications using the curve of centroids.

The general idea is that a transformation is applied on the centroids of the selected cross sections. As we described in chapter 6, the transformations have the form of a homogeneous transformation matrix. Rotations, translations, scaling are simply performed by multiplying each selected centroid with the corresponding transformation matrix. The transformation matrix is built into the data structure of the cross sections, so each cross section has its own transformation matrix and all elements of the cross section (cloud points, feature points, control points, etc) are subject to it.

The form of the transformation matrix can be either very simple, or very complex, with several variations and combinations of other simple transformations. Simple transformations may include rotations, translations or scaling. For example, a rotation around the x axis, a scaling, or a translation, would require us to apply the transformations R_x, S and T respectively. For other types of transformations the reader may refer to several Computer Graphics Handbooks, e.g. [171].

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7.3.2 Alignment Operator

First of all, we define the *alignment* operator, which reorganizes the slices according to the direction of the curve of centroids. The operator is applied to one or more selected cross sections, and its function is to rotate the selected slices around their corresponding centroids, so that the normal of the plane defined by the slice points coincides with the gradient direction of the curve of centroids at each centroid. This operator is performed by applying an affine transformation matrix $R_L(\theta)$ that rotates the slice points around their centroid towards the gradient direction L of the curve of centroids:

$$R_L(\theta) = T(\vec{C_i}) \cdot A(\vec{v})^{-1} \cdot R_z(\theta) \cdot A(\vec{v}) \cdot T(-\vec{C_i}) \text{ where } A(\vec{v}) = R_y(-\theta_2) \cdot R_x(\theta_1).$$

The transformation includes a translation $T(-\vec{C_i})$ of the centroid C_i to the axis origin, followed by an alignment $A(\vec{v})$ of the gradient vector $\vec{v}$ at C_i with the identity vector $\vec{k}$ on the z-axis, then a rotation $R_z(\theta)$ of angle θ around the z-axis to align it to the desired direction L, followed by an alignment $A(\vec{v})^{-1}$ of the identity vector $\vec{k}$ on the z-axis back to the gradient vector $\vec{v}$, and a translation $T(\vec{C_i})$ of the axis origin back to the centroid C_i. Such a transformation is standard for aligning an object to a specified direction, and is typically encountered in Graphics handbooks (e.g. examples 3.13, 3.14 in [171]).

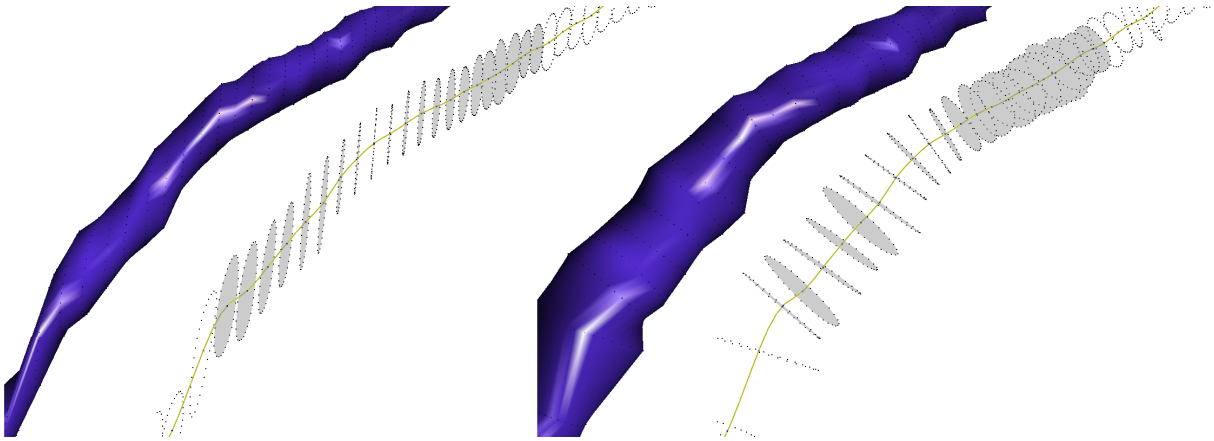


Figure 7.3: Alignment of the selected cross sections to the curve of centroids.

This operator is essential when editing the model, as it can improve the quality of the modifications or correct possible errors that occur during the editing process. It could even be used in cases where the slicing direction of the input model has not been satisfactory for the entire model. As one may notice in Figure 7.3(left), the

initial positioning of the slices does not comply with the direction of the curve of centroids as required. We can restore their positioning by aligning them to the curve as illustrated in Figure 7.3(right).

However, with a close observation on the resulting model, one would notice that the volume is not preserved with this deformation. This is a violation to the user requirements in most cases. However, the parameter we are interested in is the connectivity among the slices. For this reason, we did not concern about volume preservation in our demo implementation. Of course, in case someone would like to release an application with a full implementation, obviously the volume preservation of the model would be an essential requirement.

The editing operators described in the following section can be applied either in conjunction with the alignment operator, or without it. In case they are applied exclusively, the slices of the modified region will remain parallel after the transformations.

7.3.3 Displacement Operator

The *displacement* operator, as its name states, performs a displacement on a selected part of the model, and the connection between the modified part and the rest of the model is updated properly, to preserve connectivity and smoothness to all parts of the model. First of all, one or more cross sections have to be selected. Then, a series of transformations are applied to the selected centroids, causing the selected region to change in shape. As the curve of centroids changes in shape, the surface of the model also changes, to preserve its relative position to the curve of centroids. The modification may be an arbitrary user-specified transformation.

As illustrated in Figure 7.4 the user has selected some centroids (from C_1 to C_2) and translates them upwards on the z-axis. To maintain a smooth connectivity at the endpoints, a number of centroids outside the selected region is also modified (i.e. C_0 to C_1 and C_2 to C_3), applying a proportional transformation which brings these centroids closer to the modified endpoints. The number of centroids affected by a transformation on a selected region is specified by the user as a parameter during the editing process.

The alignment operator has also been applied to the slices with centroids from C_0 to C_1 and from C_2 to C_3. Algorithm 7.1 describes the details of the *displacement* operator.

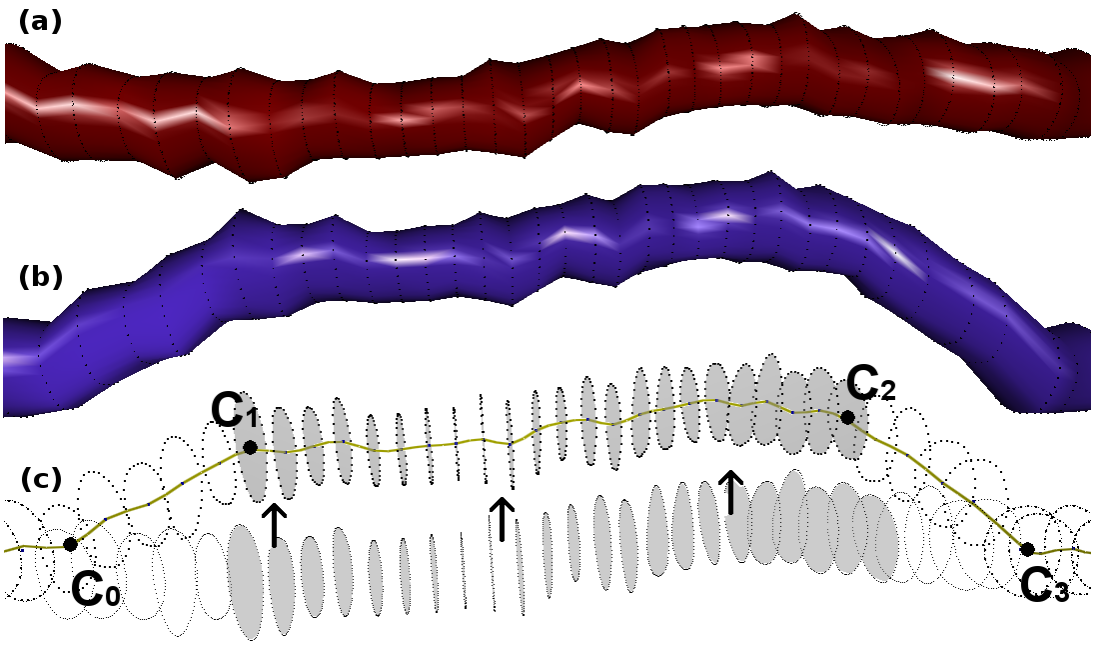


Figure 7.4: Translation using the displacement operator. (a) The initial model in red, (b) the edited model in blue, (c) details of the operator.

Algorithm 7.1 The algorithm for the Displacement Operator.

Input: Centroids C_1, C_2

Translation vector $v \leftarrow [t_x, t_y, t_z]$

Parameter n (number of centroids affected outside selected region)

for each centroid C_i from C_0 to C_1 **do**

$n \leftarrow |C_1 - C_0|$

$h \leftarrow \frac{i}{n}$

$C_i' \leftarrow C_i h v^T$ (proportional translation)

end for

for each centroid C_i from C_1 to C_2 **do**

$C_i' \leftarrow C_i v^T$ (translation)

end for

for each centroid C_i from C_2 to C_3 **do**

$n \leftarrow |C_3 - C_2|$

$h \leftarrow 1 - \frac{i}{n}$

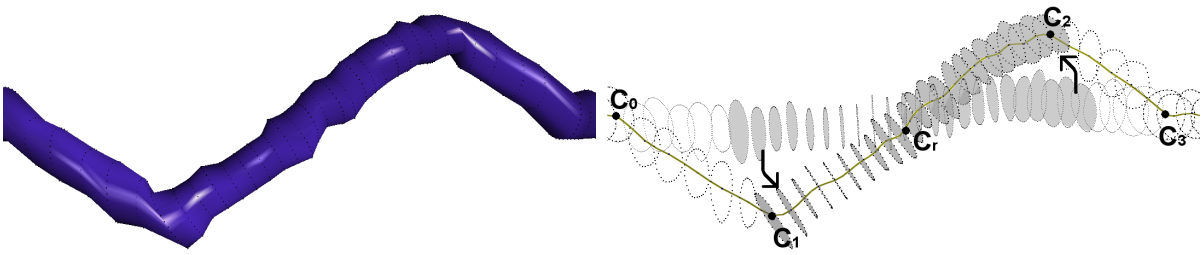
$C_i' \leftarrow C_i h v^T$ (inverse proportional translation)

end for

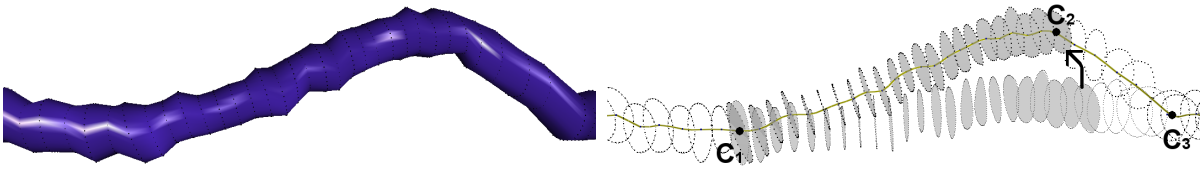
The displacement operator does not only work for translation, but also for rotation and scaling, and any arbitrary transformation. All that is needed is to replace the translation vector v by another transformation matrix R or S. For example, a rotation of angle θ around the z-axis with a point of reference C_r would be

$$R = T(\vec{C_r}) \cdot R_z(\theta) \cdot T(-\vec{C_r})$$

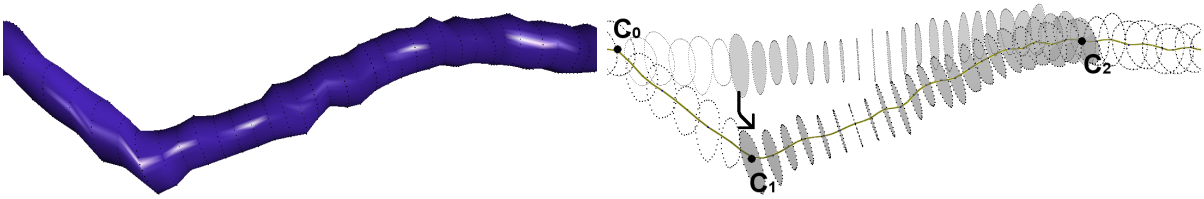
In the examples of Figure 7.5 we have three rotations around the axis defined by the gradient of the curve of centroids at the reference point, i.e. the center of the rotation. In Figure 7.5(a) the point of reference is $C_r = (C_1 + C_2)/2$, while in Figures 7.5(b) and 7.5(c) it is $C_r = C_1$ and $C_r = C_2$ respectively. The point of reference is not affected by the rotation, so in the cases where one of the endpoints is the point of reference, there is no need to modify any cross sections outside the region at the fixed endpoint.



(a) The selected region is rotated around $C_r = (C_1 + C_2)/2$, on the axis defined by the gradient of the curve of centroids at C_r.



(b) The selected region is rotated around C_1, on the axis defined by the gradient of the curve of centroids at C_1. The cross sections left of C_1 are not affected by the transformation.



(c) The selected region is rotated around C_2, on the axis defined by the gradient of the curve of centroids at C_2. The cross sections right of C_2 are not affected by the transformation.

Figure 7.5: Rotations using the displacement operator.

7.3.4 Elastic Operator

This operator modifies the selected part of the model like an elastic rubber-band that is being stretched. This operation is applied to a selected part of the model, from centroid C_1 to C_2, causing the selected region to become tight and approximate the line segment defined by the endpoints to a certain extent. The selected centroids are translated by a parameter α that represents the amount of force being applied to the selected region, causing it to stretch. This parameter is corresponds to the tension on the curve, and is expressed as a percentage that defines how much closer the curve of centroids will get to the segment of endpoints. Tension 0% means the curve remains

unaffected, while 100% tension means that the curve of centroids degenerates to the line segment defined by the two endpoints. Figure 7.6 offers a visual explanation of the parameter α and algorithm 7.2 shows the formula which brings the centroids to their final positions. Figure 7.7 shows a region being stretched at 25%, 50%, 75% and 100%.

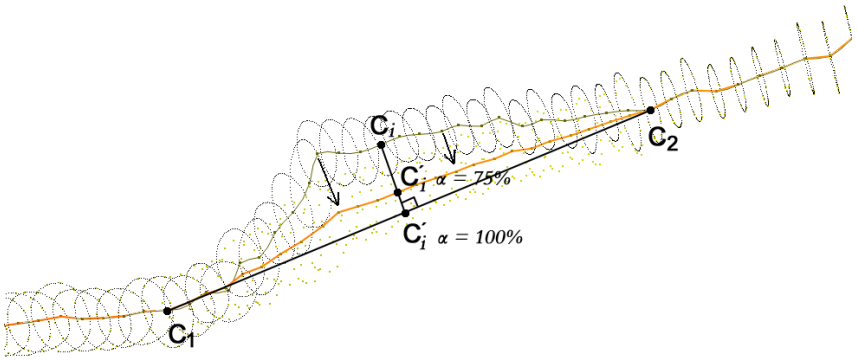


Figure 7.6: The centroid C_i is getting closer to the line segment $[C_1, C_2]$, as the parameter α increases. For $\alpha = 0$ we have $C_i' = C_i$, and for $\alpha = 100\%$ the centroid C_i is projected on the line segment $[C_1, C_2]$.

Algorithm 7.2 The algorithm for the Elastic Operator.

Input: Centroids C_1, C_2
Parameter α (tension %)

for each centroid C_i from C_1 to C_2 **do**
$$C_i' \leftarrow (1-\alpha)C_i + \alpha C_1 + \alpha(C_2 - C_1)\frac{(C_2 - C_1)\cdot(C_i - C_1)}{||C_2 - C_1||}$$
end for

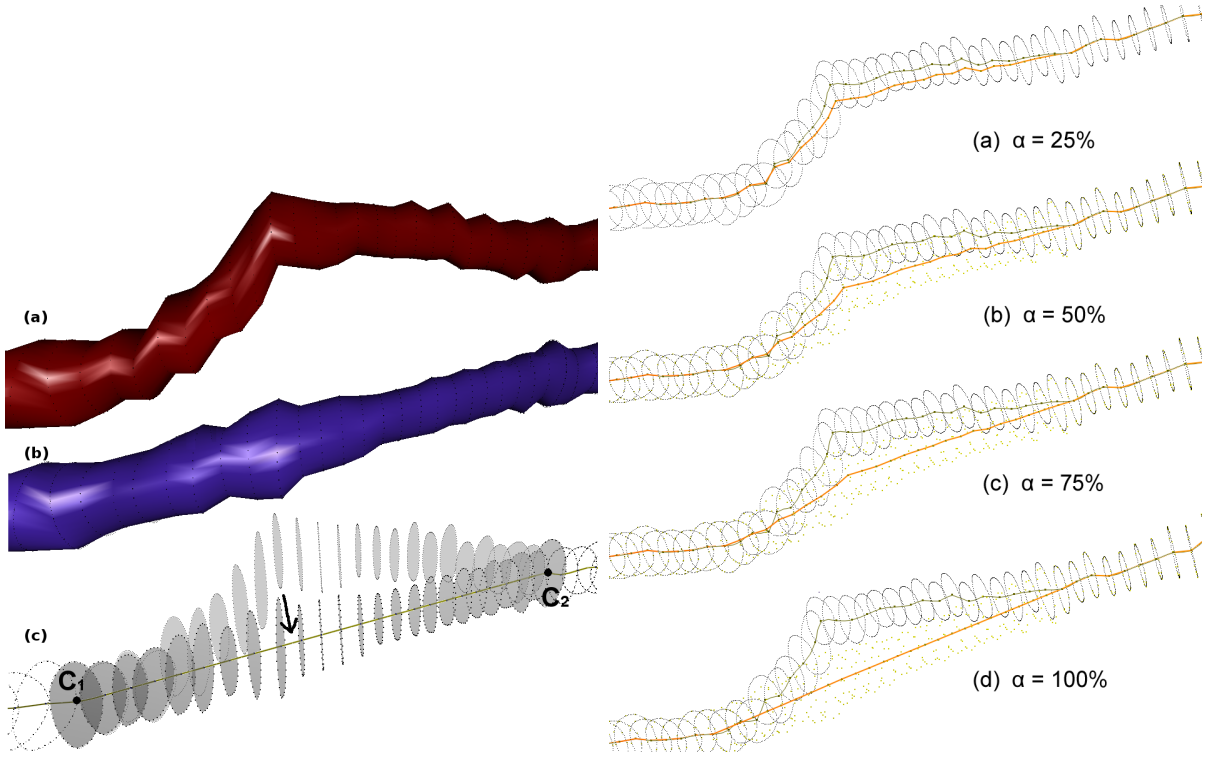


Figure 7.7: (Left) The initial model (a), the edited model (b) and the transformation (c). (Right) Applying 25%, 50%, 75% and 100% tension to the selected centroids.

7.3.5 Inflation Operator

This operator modifies the selected part of the model like a balloon being inflated. In this case the selected centroids are not affected by the transformation, but the slice points of each cross section are subjected to a scaling transformation around their centroids. Again, to preserve a smooth transition between the cross sections inside and outside the selected region, we set a number of cross sections outside the end points to follow the transformation proportionally.

Except for the percentage of scaling, we define two more parameters, i.e. the number of cross sections affected outside C_1 and C_2. This time we have two parameters with different values, as the number of cross sections affected by the proportional transformation may be different in the two sides of the selected region. Figure 7.8 shows an example of the inflation operator and algorithm 7.3 describes the process of applying the operator to the selected centroids. Centroids C_0, C_1, C_2 and C_3 are specified as parameters by the user, along with the scaling factor.

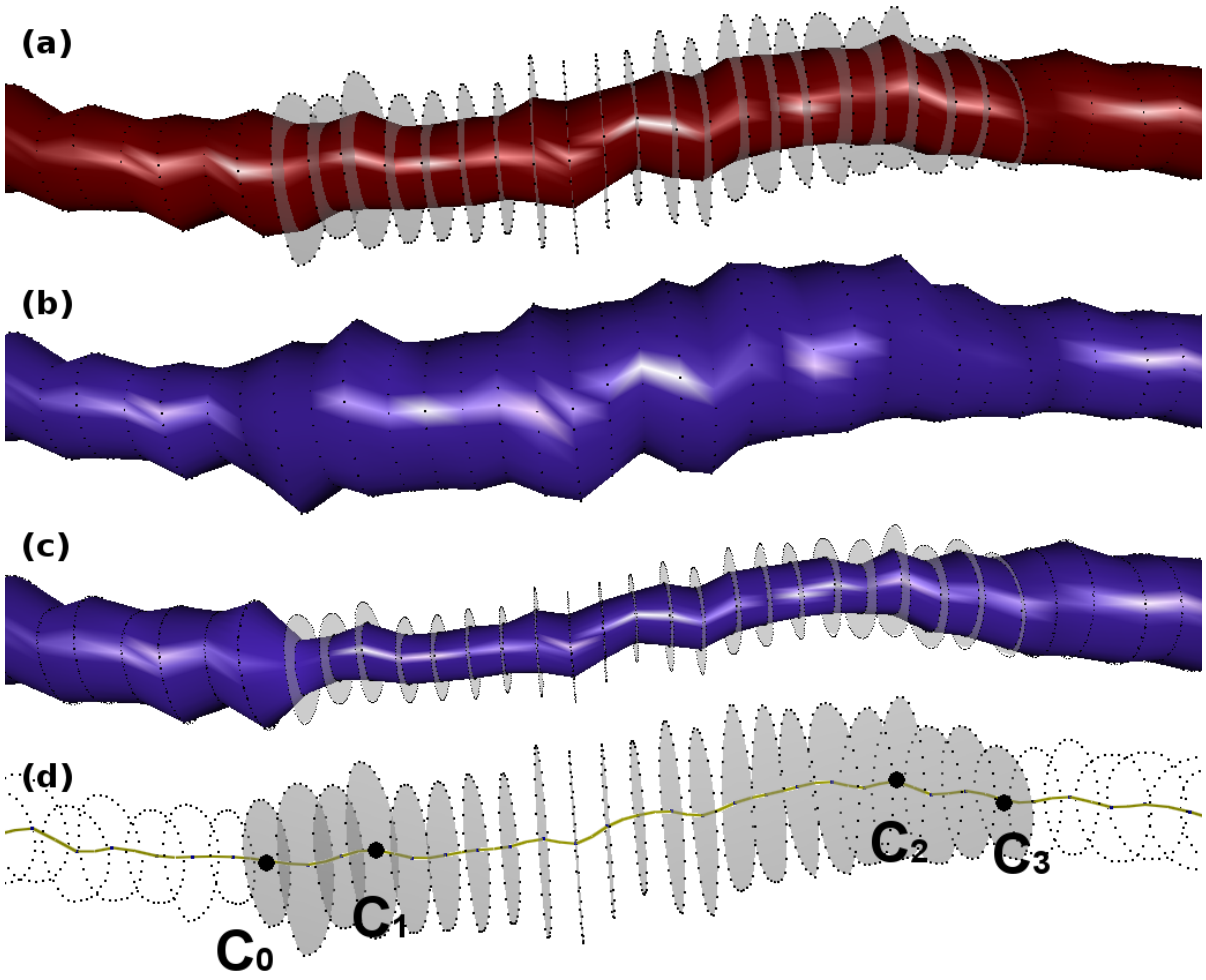


Figure 7.8: The curve of centroids remains unaffected by the transformation, but the points of the selected cross sections are scaled around their centroids. (a) The initial model, with the selected cross sections inflated. (b) The inflated model. (c) For $s < 1$ the operator causes deflation. (d) The transformation. From centroid C_0 to C_1 and from C_2 to C_3 , scaling is applied proportionally. From C_1 to C_2 the cross sections are completely scaled.

Algorithm 7.3 The algorithm for the Inflation Operator.

Input: Centroids C_1, C_2
Scaling factor s
Parameters: Centroids C_0, C_3 (centroids affected outside selected region)

for each centroid C_i from C_0 to C_1 **do**
  $n_1 \leftarrow |C_1 - C_0|$, $h_1 \leftarrow \frac{i}{n_1}$
  for each point P_j in cross section i **do**
    $P'_j \leftarrow P_j h_1 s$ (proportional scaling)
  end for
end for
for each centroid C_i from C_1 to C_2 **do**
  for each point P_j in cross section i **do**
    $P'_j \leftarrow P_j s$ (full scaling)
  end for
end for
for each centroid C_i from C_2 to C_3 **do**
  $n_2 \leftarrow |C_3 - C_2|$, $h_2 \leftarrow 1 - \frac{i}{n_2}$
  for each point P_j in cross section i **do**
    $P'_j \leftarrow P_j h_2 s$ (inverse proportional scaling)
  end for
end for

where the scaling factor s around the centroid C_r could be expressed as a transformation matrix

$$s = T(\vec{C_r}) \cdot S \cdot T(-\vec{C_r})$$

7.3.6 Placing the curve of centroids on a parametric function

Another type of operator, which is based on a parametric function, a feature previously described in section 6.2, applies a transformation that causes the cross sections to follow the path of a function or a user specified mathematical expression. This option produces impressive results with a little effort, as a single transformation may provide

different modifications to the cross sections, depending on their index in the selected region. Figure 7.9 shows such a modification, where the model of the artery has been placed on a sine function $f(h) = sinh$. Algorithm 7.4 demonstrates how to use the parametric function for the operator.

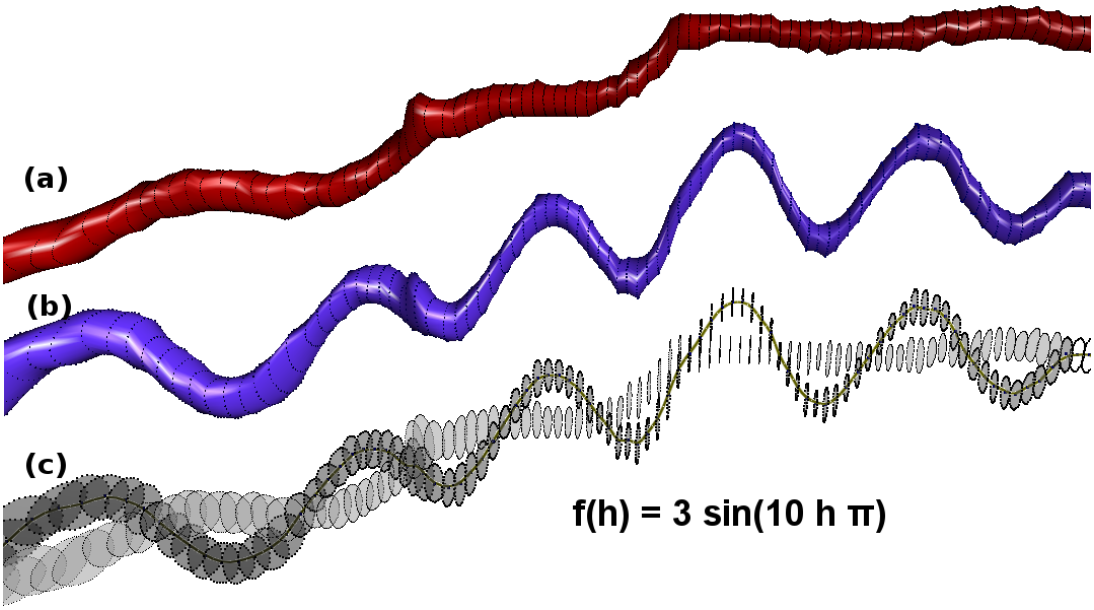


Figure 7.9: Editing the cross sections with a parametric function. Each cross section is positioned on a path defined by a mathematical expression according to its index in the selected region. Here the curve of centroids is placed on a sine function. (a) Initial model, (b) edited model, (c) the transformation.

Algorithm 7.4 The algorithm for the Parametric Function Operator.

Input: Centroids C_1, C_2
Parameter: Function $f(h)$

for each centroid C_i from C_1 to C_2 **do**
$n \leftarrow |C_2 - C_1|$, $h \leftarrow \frac{(i-1)}{(n-1)}$
$C_i' \leftarrow C_i f(h)$ (parametric function)
end for

The user has also the option to define his own curve of centroids, by choosing a set of feature points on a 3D space, and then interpolate a curve to these points. The

existing curve of centroids is placed on top of the user defined curve and the cross sections are positioned accordingly, causing the model to deform in the shape the user has sketched. The relative positions of the centroids on the new curve are calculated using the arc length [77, 78, 172], which places the first centroid on the beginning of the curve, the last centroid on the end of the curve, and all centroids in between, according to their relative length from the old curve to the new. In the example of Figure 7.10, the user has defined a set of feature points on R^3, and computed a curve that interpolates these points in the form of a helix. The artery model was then placed on this helix curve of centroids, causing the entire model to stretch in the shape of the helix.

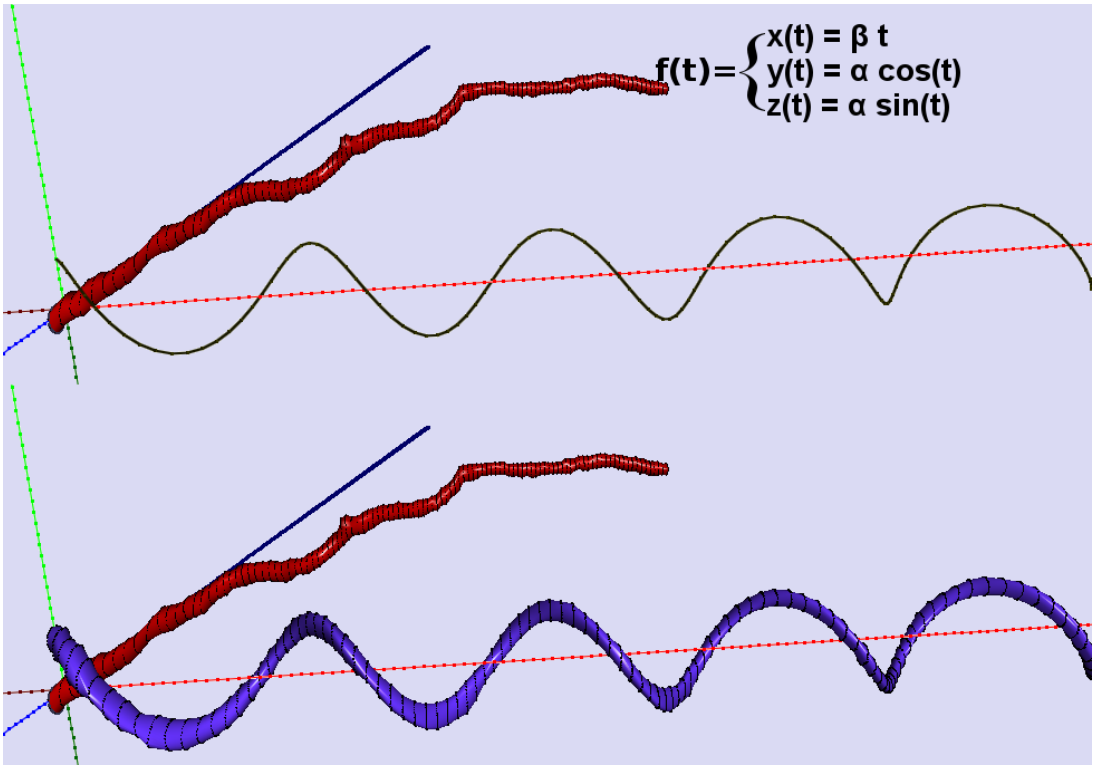


Figure 7.10: (Up) The user defines a set of feature points to form a helix in R^3 and interpolates a NURBS curve on these points. (Down) The curve of centroids of the model is placed on this curve, causing the model to deform in the shape of the helix.

7.3.7 Cross-Sectional Free-Form Editing

A more generalized and unrestricted variation of the editing operator is the one that applies modifications to the model without any consideration about the curve of centroids. So far, when a transformation was applied, the reference points for the

transformation would always be the centroids of the selected cross sections, or one point calculated from the selected centroids when many cross sections are selected. But the user is also allowed to choose an arbitrary point of reference to apply a transformation, to obtain different results. One simple idea would be to apply a transformation around the axis origin, or around a specified point on a particular axis. The surface points of the selected cross sections would then be modified in a different shape, providing another resulting model. This operator can be used to refine local details or correct any flaws on the surface of the resulting model. An operator like that was described in chapter 6, where the models where not from medical datasets, but free form objects. The variation we discuss here includes the option to use any point of reference as center for the transformation, and not only the axis origin, or the centroid of a cross section.

7.3.8 Limitations

The editing operators we described in this chapter may be used in free-form editing of medical datasets, such as arteries, producing models of any possible shape and properties. The curve of centroids is an important feature that is computed from the surface points of the initial model, and when it is modified, it defines the shape of the resulting model. The displacement, elastic and inflation operators make use of this feature, and when applied to a model, they produce models that simulate the behavior of real arteries in a surgical operation. The cross sectional free form editing operator can be used for arbitrary modifications without any consideration for the curve of centroids, or for local refinements when the modifications applied previously have caused connectivity or smoothness issues on the model.

In all cases in which we applied our editing operators, there was one specific parameter that remained fixed, i.e. the slicing direction. If we were to apply the exact same transformations on the exact same model, but with a different slicing direction, obviously the results would be also different. For example, a scaling transformation around the centroid of a cross section would result in scaling on the $x - axis$ if the slicing direction is horizontal, whereas if the slicing direction is vertical, the same transformation would result in a scaling on the $y - axis$. However, for every transformation on a given direction, there is an equivalent transformation that will offer the same result on any other direction.

The restrictions when applying these operators concern the resulting models rather than the actual modifications. In case of modeling human tissues, the physical properties of these tissues have to be preserved during the editing process. This means that a model cannot be modified in such a way that would deform a tissue beyond its limits. In other words, even if an arbitrary transformation can be applied on the model, the resulting model has to satisfy the user intentions, and generally have a useful meaning. In this work, however, we do not make any suggestions to comply with such restrictions. Additional research is required for this issue, which also includes the collaboration with medical experts.

7.4 Implementation and Examples

The editing operators we described in this chapter have also been implemented and tested under the Microsoft Visual C++ programming environment [164] using the Qt UI framework [165]. Figure 7.11 illustrates parts of the user interface that were not described in chapter 6. The following description concerns the current values and parameters of the textboxes and other components of the UI in figure 7.11.

The pop-up window in the center of the screen-shot (a) shows the options for slicing the point cloud, in case the initial point cloud did not come from a tomography scan. If the input was already in the form of cross sectional slices, we only use this menu for selecting or deselecting the regions of interest, to apply our modifications. In the example of figure 7.11, the artery model consists of 110 slices of 80 feature points each, and the cross sections from slice 58 to slice 74 have been selected.

The menu on the right side of the screen-shot (b) shows the options for applying the editing operators on the selected centroids or cross sections. The transformation matrix visible on top of the menu corresponds to a rotation matrix $R_z(\theta)$ that would rotate the selected region around the z-axis for $\theta = 30^o$ ($cos(\theta) = 0.867, sin(\theta) = 0.5$).

This rotation is performed in respect to the selected reference point, which is the centroid of slice 66, as shown bellow the slider at the displacement operator. On the left of the "Apply" button at the displacement operator, the $[C_0..C_1]$ option is the parameter of the cross sections that are affected by the transformation, on both sides of the selected region ($[C_0..C_1]$ and $[C_2..C_3]$). In this example 5 more cross sections are affected by the transformation on each side.

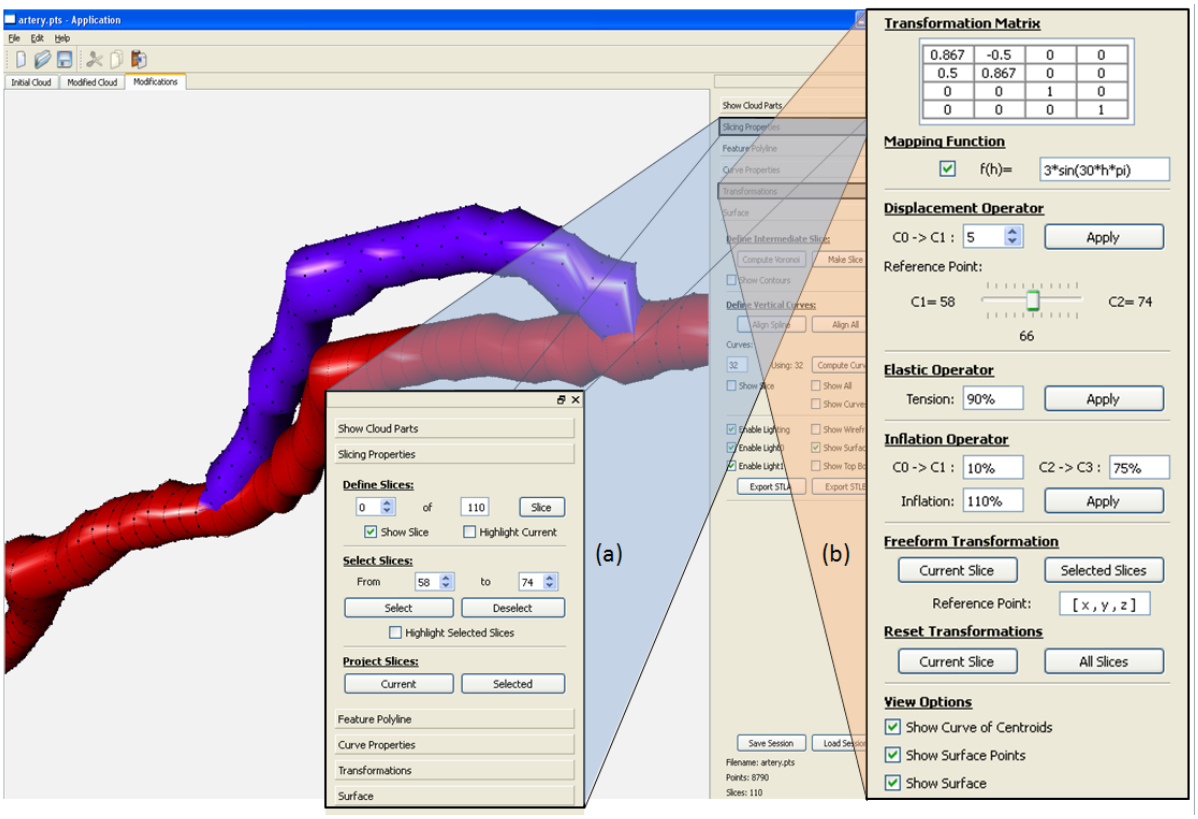


Figure 7.11: The user interface.

The elastic operator shown bellow has only one parameter, the amount of tension applied to the selected region. This value is expressed as a percentage (here 90%) that will bring the selected centroids 90% closer to the line segment $[C_1, C_2]$. Next is the inflation operator, where the parameters are the amount of cross sections inflated proportionally and the inflation factor. The numbers mean that if the entire region (from C_0 to C_3) is 100% in length, the centroid C_1 will be the one closest to 10% of length, and the centroid C_2 will be the one closest to 75% length. The cross sections corresponding to centroids from C_1 to C_2 will inflate 110%.

The free-form operator can be applied either on the currently selected slice, or the entire region of interest. The transformation matrix is applied to the surface points, but this time the reference point is not the centroid of each slice, but a user specified point in the 3-dimensional space. Also, if the option for the mapping function bellow the transformation matrix is enabled, the transformation will be different for each slice in the selected region, according the evaluation of the mapping function for the respective slice. The remaining options are for resetting previously applied transformations, and to show or hide the elements of the model.

An example of a medical application in which our editing operators could be used is the bypass surgery [173], where a segment from an artery or vein from elsewhere in the patient's body is grafted to the coronary arteries to bypass atherosclerotic narrowings and improve the blood supply to the coronary circulation supplying the heart muscle. When a doctor performs such a surgery, they need to calculate the path of the artery in advance, before the segment is placed. With our editing operators, a possible path can be calculated by modifying the curve of centroids according to the doctors instructions.

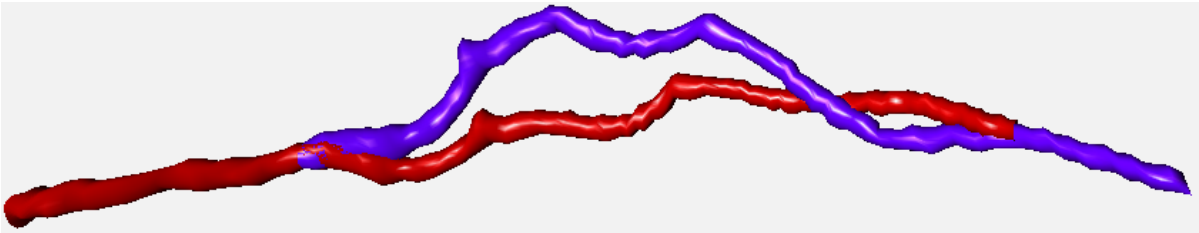


Figure 7.12: The path of an artery can be redefined by editing the curve of centroids. This is a key feature for medical applications such as the bypass surgery.

Another example of medical application in which our editing operators could be used is the angioplasty [174], a technique of mechanically widening narrowed or obstructed arteries, the latter typically being a result of atherosclerosis. An empty and collapsed balloon on a guide wire, known as a balloon catheter, is passed into the narrowed locations and then inflated to a fixed size. The balloon crushes the fatty deposits, opening up the blood vessel for improved flow, and the balloon is then deflated and withdrawn. A stent may or may not be inserted at the time of ballooning to ensure the vessel remains open. An example is shown in Figure 7.13. If a stent is inserted, it would follow the same path inside the artery, as the curve of centroids, as it is a flexible object, and will bend as the walls of the artery apply pressure on it. But as the walls of the artery apply pressure on the stent, so will the stent apply pressure on the artery walls. Furthermore, as the stent is deployed, it pushes the artery walls outwards, widening the opening of the artery.

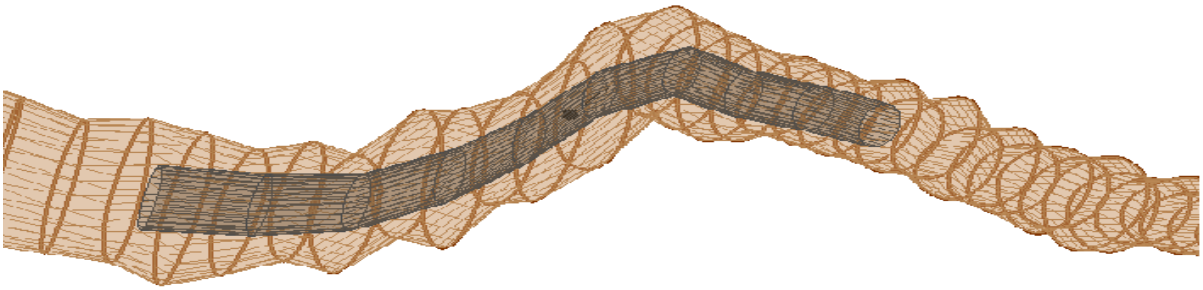


Figure 7.13: The surgical operation of angioplasty often includes a stent insertion. When a stent is inserted, the tissue of the artery is both stretched, as described with the Elastic Operator, and also inflated, as described with the Inflation operator.

The first of these two actions will modify the curve of centroids at the region where the stent is inserted, causing it to stretch to some extent (as we described with the elastic operator), as long as the flexibility of the tissue allows for stretching. The second action has no effect on the curve of centroids, but it causes the artery walls to inflate (as we described with the inflation operator). The artery will behave as the model in Figures 7.6 and 7.8.

The operators we described here could be used in medical data sets. However, this doesn't mean that they are addressed exclusively to such applications. They could be applied to other domains as well. For example, we could apply the displacement operator or the inflation operator on the drill bit model of Figure 3.10, to obtain the results illustrated in Figure 7.14. Perhaps these models have no real use for mechanical objects like a drill bit, but they are still impressive, even from a purely artistic point of view.

A video with a visual demonstration of the editing operators we described in this chapter is also available in [175].

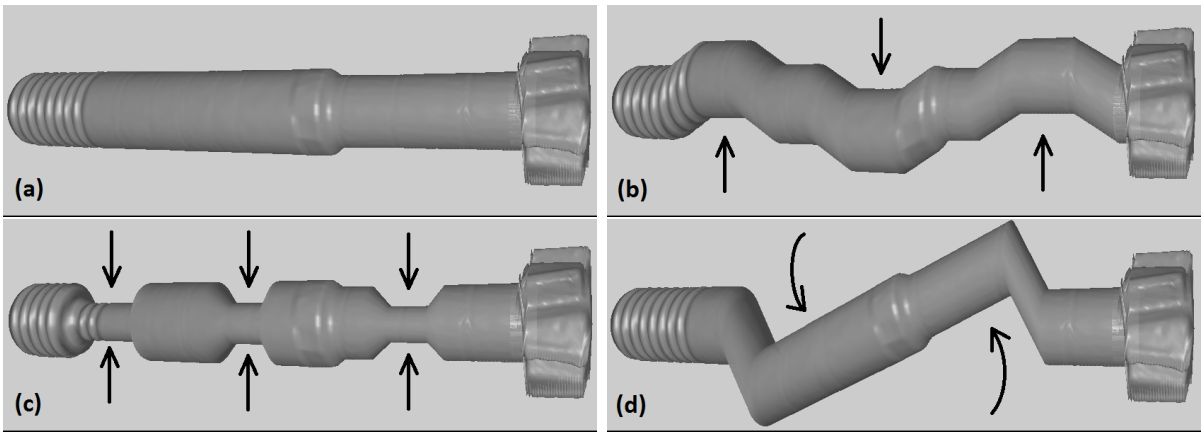


Figure 7.14: (a) The drill bit model without any modifications, (b) Using the displacement operator to translate parts of the model, (c) Using the inflation operator to deflate parts of the model, and (d) Using the displacement operator to rotate a part of the model around its center.

Chapter 8

Conclusions - Future Research Directions

8.1 Validation - Comparison to Commercial Tools

8.2 Conclusions

8.3 Future Research Directions

8.1 Validation - Comparison to Commercial Tools

To test the effectiveness of our surface reconstruction method, we have exported the resulting model from our implementation, and then imported it on several commercial solutions to compare the quality of the results. The model we used was the drill bit model of Figure 7.14, edited with a simple translation using the displacement operator as illustrated in Figure 8.1.

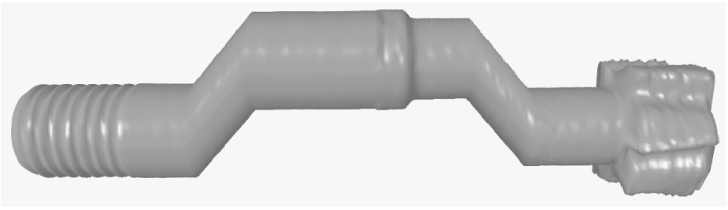


Figure 8.1: The initial drill bit point cloud consists of 1436231 points, but after applying our method, the resulting model consists of only 25600 points, which are ordered, in 200 slices x 128 points per slice.

This model was imported as a point cloud (without any structural information) to the Pointools editing solution for point clouds [176], which offers basic point cloud editing, such as point selection, organizing points in layers and RGB painting of points. Figure 8.2 shows the drill bit model, with parts of the cloud painted with different colors. This tool, however, does not offer any high level editing operators, similar to our method and does not provide surface reconstruction.

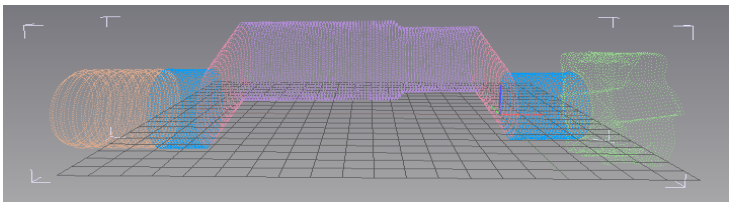


Figure 8.2: The drill bit model edited with Pointools. Parts of the point cloud have been painted to show the different parts of the model: Screw threads (orange), body (blue), proportionally displaced cross sections (pink), displaced region (purple), drill (green).

The same model was imported to the Geomagic Studio [177], which offers a variety of features, such as basic point editing (delete, sample, reduce noise), advanced point editing (add points, fill point holes, offset), basic mesh editing (delete, fill holes, trim, mesh doc), advanced polygon editing (sandpaper, patch, sculpt, shell), etc. This solution is mainly used for local editing of 3D models that come in the form of NURBS surfaces.

Unlike Geomagic, our method does not require the input to come in the form of a mesh or a CAD model. We offer editing that is applied on the points of the cloud directly, or indirectly if we use the curve of centroids for editing. The process of reconstructing the surface is independent of the editing operation, and is performed for visualization purposes only.

Moreover, the reconstruction of the model surface seems to be erroneous at some parts of the model, highlighted with arrows in Figure 8.3. Geomagic offers the proper tools for correcting such problems locally, but it requires additional processing. In our method, we utilize the structural information we have extracted, and the reconstructed surface is robust at the entire model.

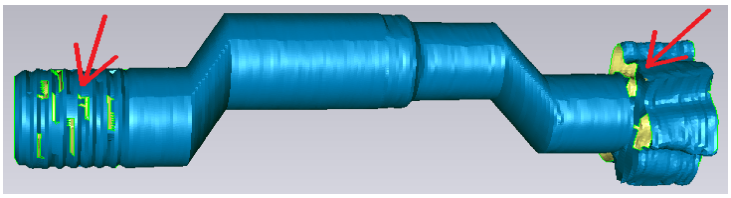


Figure 8.3: The drill bit model edited with Geomagic Studio. Some triangles at the screw threads and the drill seem to have been incorrectly computed.

To verify the quality of our surface reconstruction method, we have also imported our test model to the open source processing system MeshLab [178], where we have tried out several available methods. Our conclusion is that our method reconstructs the model surface satisfactorily, and in some cases it produces more robust results compared to other methods. For example, as illustrated in Figure 8.4, where we have reconstructed the model surface with the ball pivoting algorithm [86], the resulting model seems to have flaws in some triangles, although we have provided an ordered input.

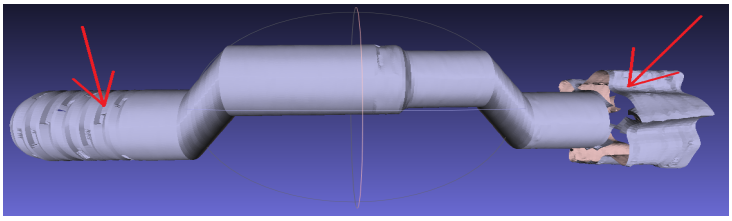


Figure 8.4: The surface of the drill bit model, reconstructed with the ball pivoting algorithm in MeshLab.

Table 8.1 sums up the comparison we made with the above-mentioned tools. As we can see in MeshLab and Geomagic studio, the offered surface reconstruction methods, do not take into account that the input may be ordered, and in some cases fail to form all triangles, leaving small holes in regions with changing curvature. On the other hand, our tool has produced a model in which the points are ordered and therefore it reconstructs the surface on the entire model, producing a watertight model.

Both MeshLab and Geomagic Studio have other options that allow for filling the holes, such as the healing tool, which correct the problems on the reconstruction step. But this requires additional effort and expert skills, and again does not consider we have an ordered input. Pointools does not offer any type of surface reconstruction,

as it only allows for grouping and coloring regions of points.

Feature \ Tool	Meshlab	Geomagic Studio	Pointools	Our Tool
Select points Group in regions	Yes (plugins)	Yes	Yes	Yes Cross-Sections
Boundary representation (feature points)	Yes (plugins)	Yes	No	Yes Convex hull Voronoi diagram
Surface reconstruction	Yes Not watertight	Yes Not watertight	No	Yes Watertight
Insert additional feature points	Yes (plugins)	Yes (healing tool)	No	Yes Convex hull Voronoi diagram
Free form editing	Yes (plugins)	Yes	No	Yes Transformation Matrix
Complex editing operators	No (manually)	No (manually)	No	Yes Curve of Centroids

Table 8.1: Comparison with other commercial tools (surface reconstruction).

Our method automates the process of selecting the points of one or more cross sections to perform modifications on these points, while it respects the initial information of the input model, which is preserved during the entire editing and reconstruction process. The $[x, y, z]$ coordinates of the point cloud are always available, as the modifications and the parameters are embedded inside the tree-form data structure of the model. This may be essential in some cases, as in medical examples, where the initial information is essential and has to be preserved during the entire process.

To validate the quality of the results of our editing tool set, we tried to apply similar modifications on our model, using 3ds Max [179]. Figure 8.5 illustrates such an example, in which we tried to apply a $sin(\theta)$ function similar to the one in Figure 6.2(d), a translation that creates the effect of a large wave. The problem in 3ds Max is that the translation is actually performed by dragging the selected control points with

the mouse. This may seem a user friendly solution, but it lacks in accuracy of the results. There is an option for setting specific transformations on the selected control points, but this option requires the individual point manipulation. In any case, 3ds Max does not offer a direct tool for applying a proportional transformation, such as the one we have developed. Therefore, our editing tool offers a powerful feature, which allows for applying parametric transformations with a single deformation.

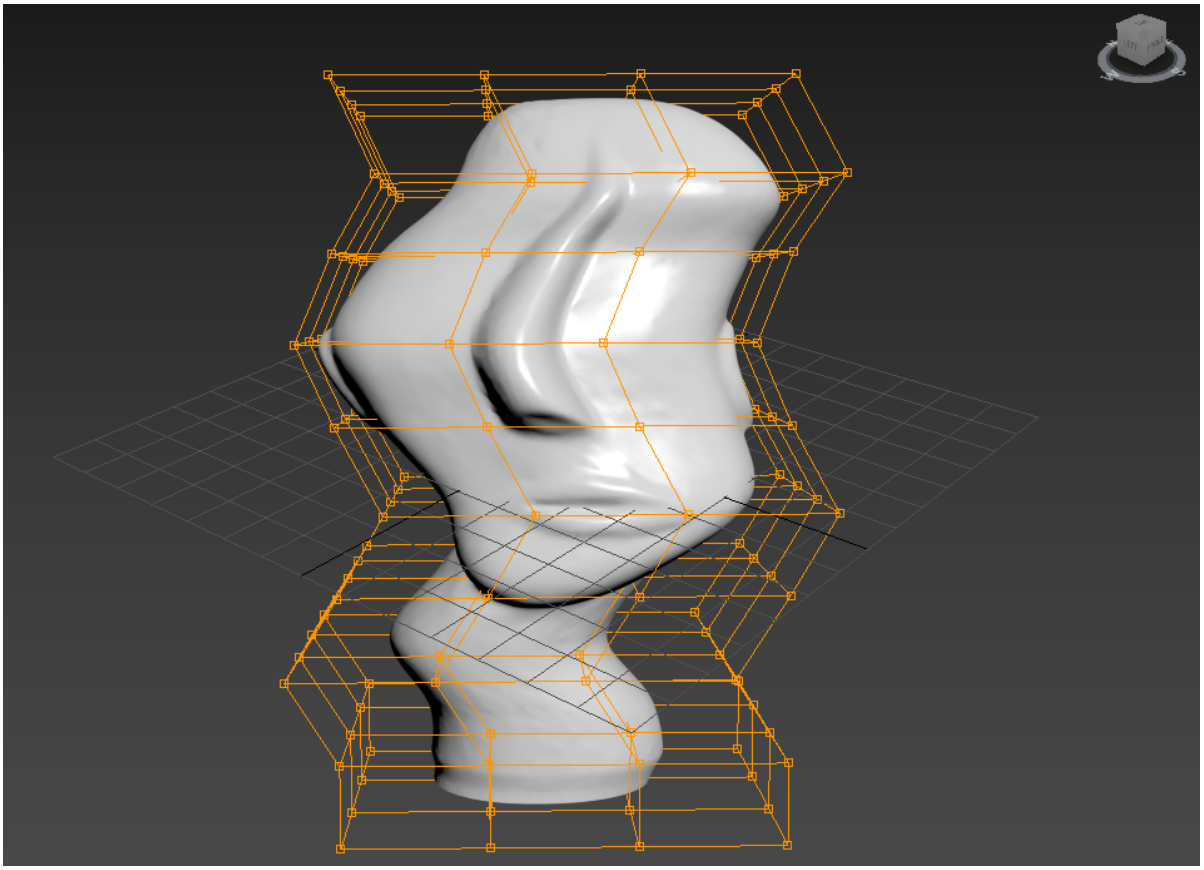


Figure 8.5: Editing with 3ds Max. Editing is performed by manually deforming individual control points on the bounding box.

Table 8.2 shows the comparison of our tool with the 3ds Max. 3ds Max is a very powerful tool, which offers several editing tools, targeted in a variety of applications. But most of the tools require editing by hand, meaning that the modifications are made by selecting a part of the model, or a set of control points (eg. on its bounding box), and by dragging the selection with the mouse. Although there is an option to set specific values on the parameters, this type of editing is considered as editing by hand, and requires additional effort and expert skills.

Moreover, 3ds Max does not directly offer complex editing operators, such as

those described in chapter 7, and certainly does not allow for editing according to a parametric function, such as we described in chapter 6.

Feature \ Tool	3ds Max	Our Tool
Requires a specific type of input	No An unstructured point cloud will do	No An unstructured point cloud will do
Selecting points Grouping in regions	Yes Manual selection with the mouse	Yes Cross-sectional slices
Use feature points	Yes Control points of bounding box	Yes B-Splines interpolating cross-sectional feature poly-lines
Allows for free form editing	Yes Manually dragging with the mouse	Yes Using a transformation matrix
Offers complex editing operators	No Editing is done by hand	Yes The curve of centroids is a powerful feature
Editing with parametric functions	No	Yes With a single operation we get impressive results

Table 8.2: Comparison with other commercial tools (editing).

8.2 Conclusions

We have introduced a framework 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.

We also have introduced a tool-set of editing operators for modifying free-form models of cross sectional datasets addressed to various applications, such as medical simulations. The modifications performed on the model resemble the deformation of the arterial tissue in a bypass or angioplastic surgery. Our editing operators are suitable for such applications, because they make use of a high level property we extracted from the sliced point cloud, i.e. the interpolating curve of centroids. This curve is a key feature on models such as arteries, which are used as input datasets in the respective medical applications.

8.3 Future Research Directions

Our current work allows the user to perform any modifications they desire arbitrarily, without any further robustness checks. In this context, a future extension is to allow imposing geometric constraints that have to be respected when applying modifications on a model.

Another issue that could be addressed more extensively is to perform a surface reconstruction benchmark [180], to acquire a quantitative comparisson of our final models to other known reconstruction methods, to determine the most appropriate method given the input we provided.

Bibliography

[1] Cyberware, "Cyberware rapid 3d scanners," 1999.

[2] E. W. Weisstein, "Correlation coefficient. from mathworld–a wolfram web resource.," *http://mathworld.wolfram.com/CorrelationCoefficient.html*.

[3] G. Renner, "Method of shape description for mechanical engineering practice," *Computers in Industry*, vol. 3, no. 1–2, pp. 137 – 142, 1982. Double Issue- In Memory of Steven Anson Coons.

[4] T. Várady, R. R. Martin, and J. Cox, "Reverse engineering of geometric models—an introduction," *Computer-Aided Design*, vol. 29, no. 4, pp. 255 – 268, 1997. Reverse Engineering of Geometric Models.

[5] W. Thompson, J. Owen, H. De St. Germain, S. Stark, and T. Henderson, "Feature-based reverse engineering of mechanical parts," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 1, pp. 57–66, 1999.

[6] P. Benkő, R. R. Martin, and T. Várady, "Algorithms for reverse engineering boundary representation models," *Computer-Aided Design*, vol. 33, no. 11, pp. 839 – 851, 2001.

[7] W. B. Thompson, H. James, H. J. de St. Germain, T. C. Henderson, and J. C. Owen, "Constructing high-precision geometric models from sensed position data," 1996.

[8] B. R. Barbero and E. S. Ureta, "Comparative study of different digitization techniques and their accuracy," *Computer-Aided Design*, vol. 43, no. 2, pp. 188 – 206, 2011.

[9] V. H. Chan, C. H. Bradley, and G. W. Vickers, "Automating laser scanning of 3d surfaces for reverse engineering," *Proc. SPIE*, vol. 3204, pp. 156–164, 1997.

[10] J. Geng, P. Zhuang, P. May, S. Yi, and D. Tunnell, "3d facecam: a fast and accurate 3d facial imaging device for biometrics applications," *Proc. SPIE*, vol. 5404, pp. 316–327, 2004.

[11] X. Peng, Z. Zhang, and H. J. Tiziani, "3-d imaging and modeling – part i: acquisition and registration," *Optik - International Journal for Light and Electron Optics*, vol. 113, no. 9, pp. 448 – 452, 2002.

[12] J. Peiró, L. Formaggia, M. Gazzola, A. Radaelli, and V. Rigamonti, "Shape reconstruction from medical images and quality mesh generation via implicit surfaces," *International Journal for Numerical Methods in Fluids*, vol. 53, no. 8, pp. 1339–1360, 2007.

[13] N. Weng, Y.-H. Yang, and R. Pierson, "Three-dimensional surface reconstruction using optical flow for medical imaging," *IEEE transactions on medical imaging*, vol. 16, no. 5, pp. 630–641, 1997.

[14] V. Stamati and I. Fudos, "On reconstructing 3d feature boundaries," *Computer-Aided Design and Applications*, vol. 5, no. 1-4, pp. 316–324, 2008.

[15] N. Amenta, M. Bern, and M. Kamvysselis, "A new voronoi-based surface reconstruction algorithm," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, (New York, NY, USA), pp. 415–421, ACM, 1998.

[16] A. G. Borş, L. Kechagias, I. Pitas, and S. Member, "Binary morphological shape-based interpolation applied to 3-d tooth reconstruction," *IEEE Transactions on Medical Imaging*, vol. 21, no. 2, pp. 100–108, 2002.

[17] G. M. Sturgeon, "Surface reconstruction from medical imaging for use in a computer-aided design (cad) environment," 2005.

[18] W. Sun, B. Starly, J. Nam, and A. Darling, "Bio-cad modeling and its applications in computer-aided tissue engineering," *Comput. Aided Des.*, vol. 37, pp. 1097–1114, Sept. 2005.

[19] T. Ju, J. Warren, J. Carson, M. Bello, I. Kakadiaris, W. Chiu, C. Thaller, and G. Eichele, "3d volume reconstruction of a mouse brain from histological sec-

tions using warp filtering," *Journal of Neuroscience Methods*, vol. 156, no. 1, pp. 84–100, 2006.

[20] A. B. Albu*, T. Beugeling, and D. Laurendeau, "A morphology-based approach for interslice interpolation of anatomical slices from volumetric images," *IEEE Transactions on Biomedical Engineering*, vol. 55, pp. 2022–2038, Aug 2008.

[21] J. Chi and C. Zhang, "Optimization of medical ct data with high precision," *Computer-Aided Design and Applications*, vol. 10, no. 1, pp. 17–31, 2013.

[22] W. Mondy, B. Rekepalli, and P. Patel, "Three-dimensional reconstruction of large micro-ct datasets of vascular corrosion cast using parallel visualization," *CAD Conference and Exhibition - CAD'12, Niagara Falls, Canada*, 2012.

[23] G. Barnett, "The role of image-guided technology in the surgical planning and resection of gliomas," *Journal of Neuro-Oncology*, vol. 42, no. 3, pp. 247–258, 1999.

[24] M. J. Jackson, C. D. Bicknell, V. Zervas, N. J. Cheshire, S. J. Sherwin, S. Giordana, J. Peiró, Y. Papaharilaou, D. J. Doorly, and C. G. Caro, "Three-dimensional reconstruction of autologous vein bypass graft distal anastomoses imaged with magnetic resonance: clinical and research applications," *Journal of Vascular Surgery*, vol. 38, no. 3, pp. 621 – 625, 2003.

[25] Y. Kuroda, M. Nakao, T. Kuroda, H. Oyama, and M. Komori, "Interaction model between elastic objects for haptic feedback considering collisions of soft tissue," *Computer Methods and Programs in Biomedicine*, vol. 80, no. 3, pp. 216 – 224, 2005.

[26] B. N. Steele, M. T. Draney, J. P. Ku, and C. A. Taylor, "Internet-based system for simulation-based medical planning for cardiovascular disease," *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, pp. 123–129, June 2003.

[27] D. Steinman, D. Vorp, and C. Ethier, "Computational modeling of arterial biomechanics: Insights into pathogenesis and treatment of vascular disease," *Journal of Vascular Surgery*, vol. 37, no. 5, pp. 1118 – 1128, 2003.

[28] C. Kelleher and R. Pausch, "Using storytelling to motivate programming," *Commun. ACM*, vol. 50, pp. 58–64, July 2007.

[29] E. F. Anderson, "Real-time character animation for computer games." 2001.

[30] K. Hui, "Free-form deformation of constructive shell models," *Computer-Aided Design*, vol. 35, no. 13, pp. 1221 – 1234, 2003.

[31] X. Fang, H. Bao, P. A. Heng, T. Wong, and Q. Peng, "Continuous field based free-form surface modeling and morphing," *Computers and Graphics*, vol. 25, no. 2, pp. 235 – 243, 2001.

[32] V. Stamati and I. Fudos, "A parametric feature-based {CAD} system for reproducing traditional pierced jewellery," *Computer-Aided Design*, vol. 37, no. 4, pp. 431 – 449, 2005.

[33] V. Cheutet, C. Catalano, J. Pernot, B. Falcidieno, F. Giannini, and J. Leon, "3d sketching for aesthetic design using fully free-form deformation features," *Computers and Graphics*, vol. 29, no. 6, pp. 916 – 930, 2005.

[34] I. Llamas, A. Powell, J. Rossignac, and C. D. Shaw, "Bender: A virtual ribbon for deforming 3d shapes in biomedical and styling applications," in *ACM Symposium on Solid and Physical Modeling (SPM*, pp. 89–99, 2005.

[35] K. Yano and K. Harada, "Reconstruction of b-spline skinning surface from generalized cylinder mesh," *The Visual Computer*, vol. 26, no. 1, pp. 31–40, 2009.

[36] S. Yuwen, G. Dongming, J. Zhenyuan, and L. Weijun, "B-spline surface reconstruction and direct slicing from point clouds," *The International Journal of Advanced Manufacturing Technology*, vol. 27, no. 9, pp. 918–924, 2006.

[37] D. Wang, O. Hassan, K. Morgan, and N. Weatherill, "Efficient surface reconstruction from contours based on two-dimensional delaunay triangulation," *International Journal for Numerical Methods in Engineering*, vol. 65, no. 5, pp. 734–751, 2006.

[38] J. Daniels and E. Cohen, *Surface Creation and Curve Deformations Between Two Complex Closed Spatial Spline Curves*, pp. 221–234. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[39] J. Chai, T. Miyoshi, and E. Nakamae, "Contour interpolation and surface reconstruction of smooth terrain models," in *Visualization '98. Proceedings*, pp. 27–33, Oct 1998.

[40] R. De Toledo, B. Levy, and J.-C. Paul, "Reverse engineering for industrial-environment cad models," in *International Symposium on Tools and Methods of Competitive Engineering, TMCE 2008*, (Izmir, Turkey), Apr. 2008.

[41] Z. Yang and Y. Chen, "A reverse engineering method based on haptic volume removing," *Computer-Aided Design*, vol. 37, no. 1, pp. 45 – 54, 2005.

[42] Y. Jun, "A piecewise hole filling algorithm in reverse engineering," *Computer-Aided Design*, vol. 37, no. 2, pp. 263 – 270, 2005.

[43] P. A. Fayolle, A. Pasko, E. Kartasheva, and N. Mirenkov, "Shape recovery using functionally represented constructive models," in *Proceedings Shape Modeling Applications, 2004.*, pp. 375–378, June 2004.

[44] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, "Multi-level partition of unity implicits," *ACM Trans. Graph.*, vol. 22, pp. 463–470, July 2003.

[45] 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, 2003. 2nd GRSS/ISPRS Joint Workshop on*, pp. 67–71, IEEE, 2003.

[46] W.-K. Jeong, K. Kähler, J. Haber, and H. peter Seidel, "Automatic generation of subdivision surface head models from point cloud data," in *In Graphics Interface 2002 Conf. Proc*, pp. 181–188, 2002.

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

[48] C. Au and M. Yuen, "Feature-based reverse engineering of mannequin for garment design," *Computer-Aided Design*, vol. 31, no. 12, pp. 751 – 759, 1999.

[49] W. Ma and J. P. Kruth, "Nurbs curve and surface fitting for reverse engineering," *The International Journal of Advanced Manufacturing Technology*, vol. 14, no. 12, pp. 918–927, 1998.

[50] 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, 1988.

[51] D. Metaxas and D. Terzopoulos, "Shape and nonrigid motion estimation through physics-based synthesis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, pp. 580–591, June 1993.

[52] J. V. Miller, D. E. Breen, W. E. Lorensen, R. M. O'Bara, and M. J. Wozny, "Geometrically deformed models: A method for extracting closed geometric models form volume data," *SIGGRAPH Comput. Graph.*, vol. 25, pp. 217–226, July 1991.

[53] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, pp. 158–175, Feb. 1995.

[54] T. McInerney and D. Terzopoulos, "Deformable models in medical image analysis: a survey," *Medical image analysis*, vol. 1, no. 2, pp. 91–108, 1996.

[55] A. Dolenc and I. Mäkelä, "Slicing procedures for layered manufacturing techniques," *Computer-Aided Design*, vol. 26, no. 2, pp. 119 – 126, 1994.

[56] R. Jamieson and H. Hacker, "Direct slicing of cad models for rapid prototyping," *Rapid Prototyping Journal*, vol. 1, no. 2, pp. 4–12, 1995.

[57] P. Kulkarni and D. Dutta, "An accurate slicing procedure for layered manufacturing," *Computer-Aided Design*, vol. 28, no. 9, pp. 683 – 697, 1996.

[58] E. Sabourin, S. A. Houser, and J. Helge Bøhn, "Adaptive slicing using stepwise uniform refinement," *Rapid Prototyping Journal*, vol. 2, no. 4, pp. 20–26, 1996.

[59] R. Hope, R. Roth, and P. Jacobs, "Adaptive slicing with sloping layer surfaces," *Rapid Prototyping Journal*, vol. 3, no. 3, pp. 89–98, 1997.

[60] K. Tata, G. Fadel, A. Bagchi, and N. Aziz, "Efficient slicing for layered manufacturing," *Rapid Prototyping Journal*, vol. 4, no. 4, pp. 151–167, 1998.

[61] J. Tyberg and J. Helge Bøhn, "Local adaptive slicing," *Rapid Prototyping Journal*, vol. 4, no. 3, pp. 118–127, 1998.

[62] K. Mani, P. Kulkarni, and D. Dutta, "Region-based adaptive slicing," *Computer-Aided Design*, vol. 31, no. 5, pp. 317 – 333, 1999.

[63] W. Ma, W.-C. But, and P. He, "Nurbs-based adaptive slicing for efficient rapid prototyping," *Computer-Aided Design*, vol. 36, no. 13, pp. 1309 – 1325, 2004.

[64] S. C. Park, "Sculptured surface machining using triangular mesh slicing," *Computer-Aided Design*, vol. 36, no. 3, pp. 279 – 288, 2004.

[65] 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, no. 4, pp. 387 – 397, 2005.

[66] S. Sikder, A. Barari, and H. Kishawy, "A new adaptive slicing approach to control cusp height of rapid prototyping parts," *Proceedings of the CAD'12 Conference and Exhibition, Niagara Falls, Canada*, 2012.

[67] S. Zhong, Y. Yang, and Y. Huang, "Data slicing processing method for re/rp system based on spatial point cloud data," *Computer-Aided Design and Applications*, vol. 11, no. 1, pp. 20–31, 2014.

[68] Y. Zhang, Y. Wong, H. Loh, and Y. Wu, "An adaptive slicing approach to modelling cloud data for rapid prototyping," *Journal of Materials Processing Technology*, vol. 140, no. 1–3, pp. 105 – 109, 2003. Proceedings of the 6th Asia Pacific Conference on materials Processing.

[69] G. Liu, Y. Wong, Y. Zhang, and H. Loh, "Modelling cloud data for prototype manufacturing," *Journal of Materials Processing Technology*, vol. 138, no. 1–3, pp. 53 – 57, 2003. {IMCC2000}.

[70] Y. Wu, Y. Wong, H. Loh, and Y. Zhang, "Modelling cloud data using an adaptive slicing approach," *Computer-Aided Design*, vol. 36, no. 3, pp. 231 – 240, 2004.

[71] N. Dyn, D. Levin, and J. A. Gregory, "A 4-point interpolatory subdivision scheme for curve design," *Computer Aided Geometric Design*, vol. 4, no. 4, pp. 257 – 268, 1987.

[72] L. Fang and D. C. Gossard, "Multidimensional curve fitting to unorganized data points by nonlinear minimization," *Computer-Aided Design*, vol. 27, no. 1, pp. 48 – 58, 1995.

[73] K. Pigounakis and P. Kaklis, "Convexity-preserving fairing," *Computer-Aided Design*, vol. 28, no. 12, pp. 981 – 994, 1996.

[74] I.-K. Lee, "Curve reconstruction from unorganized points," *Computer Aided Geometric Design*, vol. 17, no. 2, pp. 161 – 177, 2000.

[75] K. M. Liang, M. Rajeswari, and B. E. Khoo, "Free form shape representation using nurbs modeling," *Proceedings of the WSCG (Short Papers)'02*, pp. 67–73, 2002.

[76] M. A. Said, "Polyline approximation of singlevalued digital curves using alternating convex hulls," *Computer Graphics and Geometry*, vol. 4, pp. 75–99, 2002.

[77] H. Wang, J. Kearney, and K. Atkinson, "Arc-length parameterized spline curves for real-time simulation," in *In in Proc. 5th International Conference on Curves and Surfaces*, pp. 387–396, 2002.

[78] X.-D. Chen, J.-H. Yong, G.-Q. Zheng, and J.-G. Sun, "Automatic {G1} arc spline interpolation for closed point set," *Computer-Aided Design*, vol. 36, no. 12, pp. 1205 – 1218, 2004.

[79] M. A. Sabin, N. A. Dodgson, M. Sabin, and N. Dodgson, "A circle-preserving variant of the four-point subdivision scheme," in *Mathematical Methods for Curves and Surfaces: Tromsø 2004, Modern Methods in Mathematics*, pp. 275–286, Nashboro Press, 2005.

[80] P. N. Azariadis and N. S. Sapidis, "Drawing curves onto a cloud of points for point-based modelling," *Computer-Aided Design*, vol. 37, no. 1, pp. 109 – 122, 2005.

[81] N. Dyn, M. S. Floater, and K. Hormann, "A c 2 four-point subdivision scheme with fourth order accuracy and its extensions," in *Mathematical Methods for Curves and Surfaces: Tromso 2004, Modern Methods in Mathematics*, pp. 145–156, Nashboro Press, 2005.

[82] Q.-X. Huang, S.-M. Hu, and R. R. Martin, "Fast degree elevation and knot insertion for b-spline curves," *Computer Aided Geometric Design*, vol. 22, no. 2, pp. 183 – 197, 2005.

[83] J. Sánchez-Reyes and L. Fernández-Jambrina, "Curves with rational chord-length parametrization," *Computer Aided Geometric Design*, vol. 25, no. 4–5, pp. 205 – 213, 2008. Pythagorean-Hodograph Curves and Related Topics.

[84] N. Dyn, M. S. Floater, and K. Hormann, "Four-point curve subdivision based on iterated chordal and centripetal parameterizations," *Computer Aided Geometric Design*, vol. 26, no. 3, pp. 279 – 286, 2009.

[85] G. Li, L. Liu, H. Zheng, and N. J. Mitra, "Analysis, reconstruction and manipulation using arterial snakes," *ACM Trans. Graph.*, vol. 29, pp. 152:1–152:10, Dec. 2010.

[86] 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, Oct. 1999.

[87] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Trans. Graph.*, vol. 13, pp. 43–72, Jan. 1994.

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

[89] S. Kels and N. Dyn, "Reconstruction of 3d objects from 2d cross-sections with the 4-point subdivision scheme adapted to sets," *Computers & Graphics*, vol. 35, no. 3, pp. 741 – 746, 2011. Shape Modeling International (SMI) Conference 2011.

[90] P. Pal, "A reconstruction method using geometric subdivision and nurbs interpolation," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 3, pp. 296–308, 2008.

[91] L. Liu, C. Bajaj, J. Deasy, D. A. Low, and T. Ju, "Surface reconstruction from non-parallel curve networks," *Computer Graphics Forum*, vol. 27, no. 2, pp. 155–163, 2008.

[92] V. Sangveraphunsiri and K. Sritrakulchai, "The development of a technique for 3d complex surface reconstruction from unorganized point cloud," *The International Journal of Advanced Manufacturing Technology*, vol. 33, no. 7, pp. 772–781, 2007.

[93] P. Pal and R. Ballav, "Object shape reconstruction through nurbs surface interpolation," *International Journal of Production Research*, vol. 45, no. 2, pp. 287–307, 2007.

[94] J.-D. Boissonnat and P. Memari, "Shape reconstruction from unorganized cross-sections," in *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, (Aire-la-Ville, Switzerland, Switzerland), pp. 89–98, Eurographics Association, 2007.

[95] X. Yang, "Surface interpolation of meshes by geometric subdivision," *Computer-Aided Design*, vol. 37, no. 5, pp. 497 – 508, 2005. Geometric Modeling and Processing 2004.

[96] Y. Ohtake, A. Belyaev, and H.-P. Seidel, "An integrating approach to meshing scattered point data," in *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, SPM '05, (New York, NY, USA), pp. 61–69, ACM, 2005.

[97] J.-H. Lee, "Modeling generalized cylinders using direction map representation," *Comput. Aided Des.*, vol. 37, pp. 837–846, July 2005.

[98] C.-C. Kuo and H.-T. Yau, "A delaunay-based region-growing approach to surface reconstruction from unorganized points," *Computer-Aided Design*, vol. 37, no. 8, pp. 825 – 835, 2005. {CAD} '04 Special Issue: Modelling and Geometry Representations for {CAD}.

[99] A. Karniel, Y. Belsky, and Y. Reich, "Decomposing the problem of constrained surface fitting in reverse engineering," *Computer-Aided Design*, vol. 37, no. 4, pp. 399 – 417, 2005.

[100] H.-Y. Tam, H.-W. Law, and H. Xu, "A geometric approach to the offsetting of profiles on three-dimensional surfaces," *Computer-Aided Design*, vol. 36, no. 10, pp. 887 – 902, 2004.

[101] R. Sviták and V. Skala, "A robust technique for surface reconstruction from orthogonal slices," *MG&V*, vol. 13, pp. 221–233, Jan. 2004.

[102] M. Peternell and T. Steiner, "Reconstruction of piecewise planar objects from point clouds," *Computer-Aided Design*, vol. 36, no. 4, pp. 333 – 342, 2004.

[103] H.-W. Lin, C.-L. Tai, and G.-J. Wang, "A mesh reconstruction algorithm driven by an intrinsic property of a point cloud," *Computer-Aided Design*, vol. 36, no. 1, pp. 1 – 9, 2004.

[104] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral surface reconstruction from noisy point clouds," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, (New York, NY, USA), pp. 11–21, ACM, 2004.

[105] P. N. Azariadis, "Parameterization of clouds of unorganized points using dynamic base surfaces," *Computer-Aided Design*, vol. 36, no. 7, pp. 607 – 623, 2004.

[106] I. Ivrissimtzis, W.-K. Jeong, and H.-P. Seidel, "Using growing cell structures for surface reconstruction," in *Shape Modeling International, 2003*, pp. 78–86, IEEE, 2003.

[107] T. K. Dey and S. Goswami, "Tight cocone: A water-tight surface reconstructor," in *Journal of Computing and Information Science in Engineering*, pp. 127–134, ACM Press, 2003.

[108] L. A. Piegl and W. Tiller, "Surface skinning revisited," *Vis. Comput.*, vol. 18, pp. 273–283, June 2002.

[109] V. Matiukas, "A survey on methods for reconstructing surfaces from unorganized point sets," *Science – Future of Lithuania*, vol. 3, no. 1, p. 10–14, 2011.

[110] S. Kels, N. Dyn, and E. Lipovetsky, "Computation of the metric average of 2d sets with piecewise linear boundaries," *Algorithms*, vol. 3, pp. 265–275, Jun 2010.

[111] B. Whited and J. Rossignac, "B-morphs between b-compatible curves in the plane," in *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09, (New York, NY, USA), pp. 187–198, ACM, 2009.

[112] I. Braude, J. Marker, K. Museth, J. Nissanov, and D. Breen, "Contour-based surface reconstruction using mpu implicit models," *GRAPHICAL MODELS*, vol. 69, no. 2, pp. 139–157, 2007.

[113] G. Barequet and A. Vaxman, "Nonlinear interpolation between slices," in *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling*, SPM '07, (New York, NY, USA), pp. 97–107, ACM, 2007.

[114] T. Ju, J. Warren, J. Carson, G. Eichele, C. Thaller, W. Chiu, M. Bello, and I. Kakadiaris, "Building 3d surface networks from 2d curve networks with application to anatomical modeling," *The Visual Computer*, vol. 21, no. 8, pp. 764–773, 2005.

[115] Y. Huang, Z.-C. Duan, G.-L. Zhu, and S.-H. Gong, "A fast triangulation algorithm for 3d reconstruction from planar contours," *The International Journal of Advanced Manufacturing Technology*, vol. 24, no. 1, pp. 98–101, 2004.

[116] G. Barequet, M. T. Goodrich, A. Levi-Steiner, and D. Steiner, "Contour interpolation by straight skeletons," *Graphical Models*, vol. 66, no. 4, pp. 245 – 260, 2004.

[117] S. Akkouche and E. Galin, "Implicit surface reconstruction from contours," *The Visual Computer*, vol. 20, no. 6, pp. 392–401, 2004.

[118] G. Cong and B. Parvin, "Robust and efficient surface reconstruction from contours," *The Visual Computer*, vol. 17, no. 4, pp. 199–208, 2001.

[119] G. Barequet, D. Shapiro, and A. Tal, "Multilevel sensitive reconstruction of polyhedral surfaces from parallel slices," *The Visual Computer*, vol. 16, no. 2, pp. 116–133, 2000.

[120] H. Zhang and D. Song, "On valid triangulation between two polygons on parallel slices," *Journal of Liaoning University*, vol. 2, pp. 111–116, 2001.

[121] S. Andrews, *Interactive generation of feature curves on surfaces*. PhD thesis, University of Toronto, 2000.

[122] S.-W. Chae and G.-M. Lee, "Volume triangulation from planar cross sections," *Computers and Structures*, vol. 72, no. 1–3, pp. 93 – 108, 1999.

[123] J. D. Fix and R. E. Ladner, "Multiresolution banded refinement to accelerate surface reconstruction from polygons," in *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, (New York, NY, USA), pp. 240–248, ACM, 1998.

[124] M. Hong, T. W. Sederberg, K. S. Klimaszewski, and K. Kaneda, "Triangulation of branching contours using area minimization," *International Journal of Computational Geometry and Applications*, vol. 08, no. 04, pp. 389–406, 1998.

[125] J.-D. Boissonnat, "Shape reconstruction from planar cross sections," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 1, pp. 1 – 29, 1988.

[126] C. L. Bajaj, E. J. Coyle, and K.-N. Lin, "Arbitrary topology shape reconstruction from planar cross sections," *Graphical Models and Image Processing*, vol. 58, no. 6, pp. 524 – 543, 1996.

[127] J.-M. Oliva, M. Perrin, and S. Coquillart, "3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram," *Computer Graphics Forum*, vol. 15, no. 3, pp. 397–408, 1996.

[128] M. W. Jones and M. Chen, "A new approach to the construction of surfaces from contour data," *Computer Graphics Forum*, vol. 13, no. 3, pp. 75–84, 1994.

[129] T. Johnson and P. E. Livadas, "A parallel algorithm for surface-based object reconstruction," *J. Math. Imaging Vis.*, vol. 4, pp. 389–400, Dec. 1994.

[130] G. Barequet and M. Sharir, "Piecewise-linear interpolation between polygonal slices," in *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, (New York, NY, USA), pp. 93–102, ACM, 1994.

[131] S. Mann, M. Lounsbery, C. Loop, D. Meyers, J. Painter, T. DeRose, and K. Sloan, "A survey of parametric scattered data fitting using triangular interpolants," *Curve and Surface Design, H. Hagen, ed., SIAM*, pp. 145–172, 1992.

[132] S. Talele, T. Johnson, and P. E. Livadas, "Surface reconstruction in parallel," in *Parallel and Distributed Processing, 1992. Proceedings of the Fourth IEEE Symposium on*, pp. 102–106, IEEE, 1992.

[133] A. B. Ekoule, F. C. Peyrin, and C. L. Odet, "A triangulation algorithm from arbitrary shaped multiple planar contours," *ACM Trans. Graph.*, vol. 10, pp. 182–199, Apr. 1991.

[134] S. Xu and W. Lu, "Surface reconstruction of 3d objects in computerized tomography," *Computer Vision, Graphics, and Image Processing*, vol. 44, no. 3, pp. 270 – 278, 1988.

[135] N. Ketarnavaz, L. R. Simar, and R. J. Figueiredo, "A syntactic/semantic technique for surface reconstruction from cross-sectional contours," *Comput. Vision Graph. Image Process.*, vol. 42, pp. 399–409, June 1988.

[136] M. Shantz, "Surface definition for branching, contour-defined objects," *SIGGRAPH Comput. Graph.*, vol. 15, pp. 242–270, July 1981.

[137] H. Fuchs, Z. M. Kedem, and S. P. Uselton, "Optimal surface reconstruction from planar contours," *Commun. ACM*, vol. 20, pp. 693–702, Oct. 1977.

[138] E. Keppel, "Approximating complex surfaces by triangulation of contour lines," *IBM Journal of Research and Development*, vol. 19, pp. 2–11, Jan 1975.

[139] J. Andrews, H. Jin, and C. Séquin, "Interactive inverse 3d modeling," *Computer Aided Design and Applications*, vol. 9, no. 6, 2012. Presented at CAD'12.

[140] Y. Chen, Z.-Q. Cheng, J. Li, R. R. Martin, and Y.-Z. Wang, "Relief extraction and editing," *Computer-Aided Design*, vol. 43, no. 12, pp. 1674 – 1682, 2011.

[141] Y. Gingold and D. Zorin, "Shading-based surface editing," *ACM Trans. Graph.*, vol. 27, pp. 95:1–95:9, Aug. 2008.

[142] M. Wand, A. Berner, M. Bokeloh, A. Fleck, M. Hoffmann, P. Jenke, B. Maier, D. Staneker, and A. Schilling, "Interactive editing of large point clouds.," in *Eurographics Symposium on Point-Based Graphics*, pp. 37–45, 2007.

[143] J. Zimmermann, A. Nealen, and M. Alexa, "Silsketch: Automated sketch-based editing of surface meshes," in *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, SBIM '07, (New York, NY, USA), pp. 23–30, ACM, 2007.

[144] L. B. Kara, C. M. D'Eramo, and K. Shimada, "Pen-based styling design of 3d geometry using concept sketches and template models," in *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, SPM '06, (New York, NY, USA), pp. 149–160, ACM, 2006.

[145] S.-H. Yoon and M.-S. Kim, "Sweep-based freeform deformations," *Computer Graphics Forum*, vol. 25, no. 3, pp. 487–496, 2006.

[146] M. Botsch and L. Kobbelt, "An intuitive framework for real-time freeform modeling," *ACM Trans. Graph.*, vol. 23, pp. 630–634, Aug. 2004.

[147] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel, "Laplacian surface editing," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, (New York, NY, USA), pp. 175–184, ACM, 2004.

[148] K. Pekkan, B. Whited, K. Kanter, S. Sharma, D. De Zelicourt, K. Sundareswaran, D. Frakes, J. Rossignac, and A. P. Yoganathan, "Patient-specific surgical planning and hemodynamic computational fluid dynamics optimization through free-form haptic anatomy editing tool (surgem)," *Medical & biological engineering & computing*, vol. 46, no. 11, pp. 1139–1152, 2008.

[149] S. Gumhold, X. Wang, and R. MacLeod, "Feature extraction from point clouds," in *Proceedings of 10th international meshing roundtable*, vol. 2001, pp. 293–305, Citeseer, 2001.

[150] W. L. Mondy, *Data acquisition for modeling and visualization of vascular tree*. PhD thesis, 2009.

[151] G. Antini, S. Berretti, A. del Bimbo, and P. Pala, "3d mesh partitioning for retrieval by parts applications," in *2005 IEEE International Conference on Multimedia and Expo*, pp. 1210–1213, July 2005.

[152] A. Agathos, I. Pratikakis, S. Perantonis, N. Sapidis, and P. Azariadis, "3d mesh segmentation methodologies for cad applications," *Computer-Aided Design and Applications*, vol. 4, no. 6, pp. 827–841, 2007.

[153] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[154] S. Bochkanov and V. Bystritsky, "Alglib, www.alglib.net, accessed jan 2010."

[155] G. Liu, Y. Wong, Y. Zhang, and H. Loh, "Error-based segmentation of cloud data for direct rapid prototyping," *Computer-Aided Design*, vol. 35, no. 7, pp. 633 – 645, 2003.

[156] "Scanny3d, scanny 3d laser scanners," *http://www.scanny3d.com*, 2008.

[157] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, pp. 469–483, Dec. 1996.

[158] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, 3rd ed. ed., 2008.

[159] J. O'Rourke, *Computational Geometry in C*. New York, NY, USA: Cambridge University Press, 2nd ed., 1998.

[160] P. Wesseling, *An Introduction to Multigrid Methods*. John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1992.

[161] T. W. Sederberg and E. Greenwood, "A physically based approach to 2-d shape blending," *SIGGRAPH Comput. Graph.*, vol. 26, pp. 25–34, July 1992.

[162] *Dynamic Time Warping*, pp. 69–84. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[163] I. Kyriazis, I. Fudos, and L. Palios, "Extracting cad features from point cloud cross-sections," *WSCG '2009: Full Papers Proceedings: The 17th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS: University of West Bohemia Plzen, Czech Republic*, pp. 137–144, February 2009.

[164] "Microsoft, visual studio c++," *https://www.visualstudio.com*, 2008.

[165] "The qt company, qt ui framework," *http://www.qt.io/*, 2008.

[166] B. Stout, "eval.c - simple arithmetic expression evaluator," *http://www8.cs.umu.se/ isak/snippets/*, 1997.

[167] D. J. Kasik, W. Buxton, and D. R. Ferguson, "Ten cad challenges," *IEEE Comput. Graph. Appl.*, vol. 25, pp. 81–92, Mar. 2005.

[168] M. Schaap, C. T. Metz, T. van Walsum, A. G. van der Giessen, A. C. Weustink, N. R. Mollet, C. Bauer, H. Bogunović, C. Castro, X. Deng, *et al.*, "Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms," *Medical image analysis*, vol. 13, no. 5, pp. 701–714, 2009.

[169] D. Lesage, E. D. Angelini, I. Bloch, and G. Funka-Lea, "A review of 3d vessel lumen segmentation techniques: Models, features and extraction schemes," *Medical Image Analysis*, vol. 13, no. 6, pp. 819 – 845, 2009. Includes Special Section on Computational Biomechanics for Medicine.

[170] U. Jandt, D. Schäfer, M. Grass, and V. Rasche, "Automatic generation of 3d coronary artery centerlines using rotational x-ray angiography," *Medical Image Analysis*, vol. 13, no. 6, pp. 846 – 858, 2009. Includes Special Section on Computational Biomechanics for Medicine.

[171] T. Theoharis, G. Papaioannou, N. Platis, and N. M. Patrikalakis, *Graphics and Visualization: Principles & Algorithms*. Natick, MA, USA: A. K. Peters, Ltd., 2007.

[172] J. Peterson, "Arc-length parameterization of spline curves," *http://www.saccade.com/writing/graphics/RE-PARAM.PDF*.

[173] "Wikipedia, coronary artery bypass surgery — wikipedia, the free encyclopedia," *http://en.wikipedia.org/w/index.php?title=Coronary_artery_bypass_surgery*, 2012.

[174] "Wikipedia, angioplasty — wikipedia, the free encyclopedia," *http://en.wikipedia.org/w/index.php?title=Angioplasty*, 2012.

[175] I. Kyriazis and I. Fudos, "Youtube video: Editing operators for cross-sectional data sets," *https://www.youtube.com/watch?v=JeW-bhOk-GA?vq=hd720*, 2017.

[176] "Pointools. bentley pointools v8i," *http://www.pointools.com*.

[177] "Geomagic, geomagic studio," *http://www.geomagic.com*, 2012.

[178] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference* (V. Scarano, R. D. Chiara, and U. Erra, eds.), The Eurographics Association, 2008.

[179] "Autodesk 3ds max," *https://www.autodesk.com/products/3ds-max/overview*, 2016.

[180] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, "A benchmark for surface reconstruction," *ACM Trans. Graph.*, vol. 32, pp. 20:1–20:17, Apr. 2013.

[181] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*. Natick, MA, USA: A. K. Peters, Ltd., 1993.

[182] "Maple 11, mathematics, modeling, simulation," *https://www.maplesoft.com/products/maple*, 2010.

Appendix A

Rejected Approaches

A.1 Introduction

A.2 The Correlation Coefficient

A.2.1 Reason for Rejecting this Approach

A.3 The Alternating Convex Hull

A.3.1 The Internal - External Point switching effect

A.4 The Voronoi diagram based approach - Variations

A.4.1 Using the farthest Voronoi Vertex

A.4.2 Using a single Voronoi Vertex from a bounded area

A.4.3 Using multiple Voronoi Vertices

A.5 The Rational Bezier Curve Fitting Approach

A.5.1 Reason for Rejecting this Approach

A.1 Introduction

During the study and the implementation of this research, there have been several cases where an approach had been adopted, but was eventually rejected due to the insufficient quality of the results. Most of these approaches are not worth mentioning, as they were rejected on an early stage of development. But there are some approaches which were studied and developed to a significant extent, and were rejected after being fully implemented, for the reason of providing low quality results. In the next

sections we provide a brief description of these approaches, and explain the reasons for rejecting them.

A.2 The Correlation Coefficient

After slicing the point cloud and projecting the points of each cross section onto its corresponding 2D plane, we attempted a different approach for extracting the feature poly-line from the 2D points. Our first thought was not to extract a boundary representation from the points of a cross section, but to use a thinning technique to construct a closed polygon curve that accurately represents the shape implied by these points. Such a polygon curve is illustrated in figure A.1.

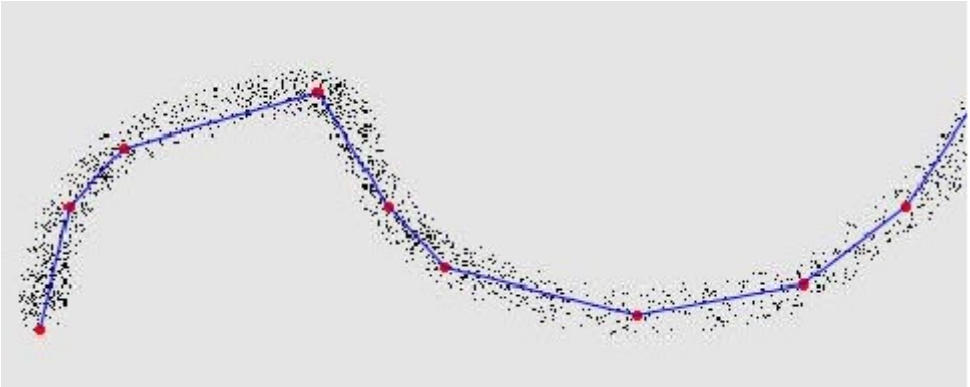


Figure A.1: A thinning technique can be applied to the 2D point set, to construct a closed polygon curve that accurately represents the shape implied by these points.

According to this approach, which was discussed in [70], to approximate the point set accurately, the polygonal curve must keep the feature points of original shape defined by points set. To compute the feature poly-line, we employ the correlation concept to determine the radius of neighborhood adaptively in the process of curve construction. Correlation refers to the degree of association between two or more quantities. In a two-dimensional plot, the correlation coefficient is used to measure the strength of the linear relationship between two variables on the two axes. Let X and Y be two variables, then the correlation coefficient of X and Y can be defined as

$$\rho(X,Y) = \frac{Cov(X,Y)}{S(X)S(Y)} \qquad \text{(eq 1.1)}$$

where $Cov(X,Y) = E[(X - E(X))(Y - E(Y)] = E(XY) - E(X)E(Y)$ and $E(\zeta)$

denotes an expectation of a random variable ζ. $S(\zeta)$ represents a standard deviation of a random variable ζ. Let (X, Y) stand for a set of N data points $P_i = (x_i, y_i)|i = 1, .., N$, then eq. eq 1.1 can be re-written as

$$\rho(X,Y) = \left| \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{N}(y_i-\bar{y})^2}\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2}} \right| \qquad \text{(eq 1.2)}$$

where $\bar{x}$ and $\bar{y}$ are the average values of x_i and y_i respectively. $\rho(X, Y)$ has a value between 0 and 1 representing the degree of linear dependence between X and Y.

This idea is used to check the linearity of the points with a neighborhood. In the problem of planar curve construction, we need to find the maximal neighborhood for each segment, in which a line segment can accurately fit the points (fig. A.2. Using this idea of correlation coefficient, we can determine the neighborhood radius adaptively.

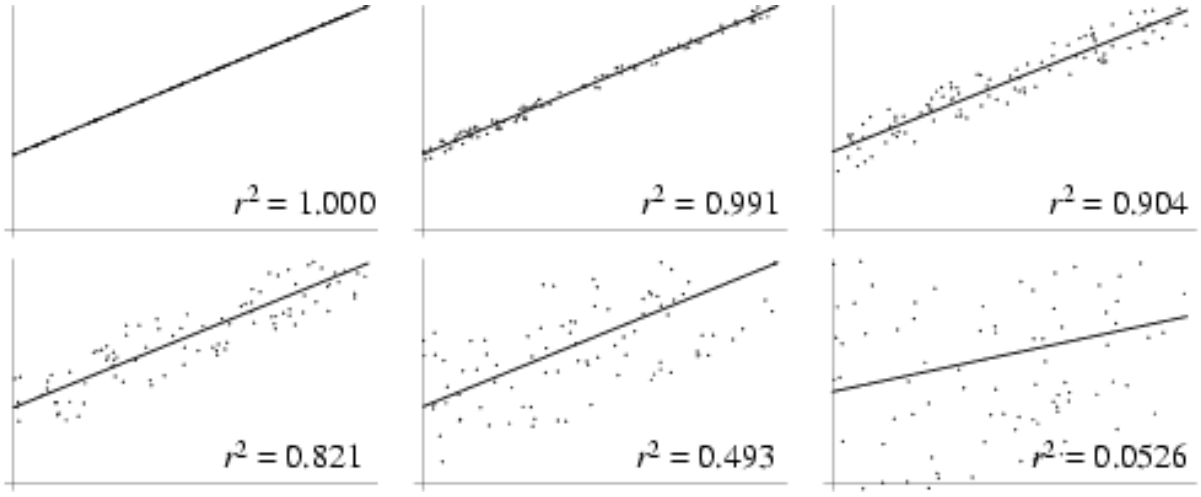


Figure A.2: Values of the correlation coefficient ρ for data sets of variable point dispersion (ρ is represented as r in the source of this figure [2]).

The initial point IP is a reference point to start the construction of the first segment of the polygonal curve from planar data points. As the points are unorganized and error-filled, the initial point selection is very important. First a start point is randomly selected, and then a fixed radius is used to find its neighborhood points. The correlation coefficient ρ of this neighborhood is then calculated. If ρ is larger than a pre-set bound, this neighborhood is used to find the initial point IP, i.e. the point that is nearest to the center of this neighborhood. Otherwise, we will re-select a point and repeat this checking process. For the case in figure A.3, point Q will be dropped due to its poor linearity, while point P can be used as the start point to find

the initial point IP. The initial point IP can then be used as a reference point for the first segment construction.

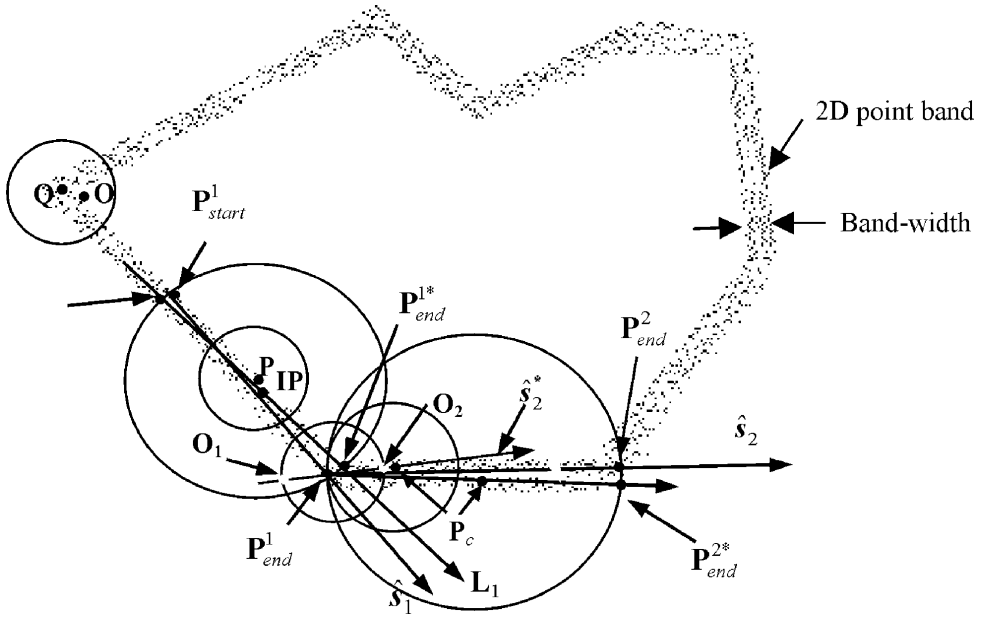


Figure A.3: Initial point IP determination and first, second segment construction.

After the initial point is identified, its neighborhood (for the first line segment, S^1) is obtained such that the ρ satisfies the user requirement. At the same time, it is necessary to make the neighborhood radius R as large as possible so that the resulted polygon has the minimum number of line segments. Hence, R needs to be determined adaptively. In our approach, we start with a conservatively small value of R and search for the close-to-optimal neighborhood radius based on the correlation coefficient. A small ρ means poor linearity and thus we need reduce the neighborhood radius; a large one means good linearity and we can increase the neighborhood radius. This iterative process for the initial point IP determination is described as follows:

Algorithm A.1 The algorithm for finding the initial point in the thinning approach using the correlation coefficient.

find_neighborhood_S^1(data set C, initial point IP, initial radius R, radius increment ΔR, correlation coefficient bounds ρ_{low}, ρ_{high})

Step 1. Select all the points P_i from C, such that $\|Pi - IP\| \leq R, P_i \in C$, to form a data set C_1.

Step 2. Compute the correlation coefficient ρ of data set C_1 using eq. eq 1.2.

Step 3.

if $\rho > \rho_{high}$ **then**

$R = R + 2\Delta R$, go to step (1).

else if $\rho < \rho_{low}$ **then**

$R = R - \Delta R$, go to step (1).

else

Return R and points from C_1, stop.

end if

With these neighborhood points having good linearity, we can construct a straight line segment that locally fits these points. Here, we use a least-square method to compute a regression line, which passes the initial point (x_{IP}, y_{IP}) and best fits the points within the neighborhood. Let $C_1 = P_i = (x_i, y_i)|i = 1, ..., N$ be the neighborhood points, a straight line, $L_1 : y = \alpha(x - x_{IP}) + y_{IP}$, can be computed by minimizing a quadratic function:

$$\varepsilon = \sum_{i=1}^{N} (\alpha(x_i - x_{IP}) + y_{IP} - y_i)^2 \qquad \text{(eq 1.3)}$$

As shown in figure A.3, line L_1 has two intersection points, P^{1*}_{start} and P^{1*}_{end}, with the neighborhood circle (centered at the initial point with a radius R). In theory, P^{1*}_{start} and P^{1*}_{end} can be considered as the start and end points of the first segment. However, they may not be among the points within the neighborhood. Thus, we select two points, which are the closest to P^{1*}_{start} and P^{1*}_{end} respectively, within the neighborhood, as the start and end points, i.e. the closest to P^{1*}_{start} as P^{1}_{start} and the closest to P^{1*}_{end} as P^{1}_{end}. S^1 is therefore obtained. $P^{1}_{start}P^{1}_{end}$ also defines the unit oriented vector of this neighborhood $(\hat{s}_1 = P^{1}_{end} - P^{1}_{start})/\|P^{1}_{end} - P^{1}_{start}\|)$. Using $P^{1}_{start}P^{1}_{end}$ as the diameter, a new neighborhood circle is obtained, we then delete all the other points within this circle. The remaining planar data set C is also updated.

The method for constructing the remaining segments is slightly different from that of the first segment. We begin with P^1_{end} as the start point for the second segment, i.e. P^2_{start}. We then adaptively determine the neighborhood for S^2. Since the same algorithm is used for constructing the remaining segments, we denote the start point as $P^i_{start}(i \geq 2)$. The algorithm to find the radius of the neighborhood of S^i is described as follows:

Algorithm A.2 The algorithm for finding the radius of the neighborhood of S^i.

---

find_neighborhood_S^i(data set C, P^i_{start}, initial radius R, radius increment ΔR, correlation coefficient bounds ρ_{low}, ρ_{high})

Step 1. Construct a neighborhood circle C centered at P^i_{start} with radius R.

Select all points P_k from C, such that $\|P_k - P^i_{start}\| \leq R$,
to form a data set $C_i(k = 1, 2, .., n)$.
Compute the correlation coefficient ρ of data set C_i.
if $\rho < \rho_{low}$ **then**
  $R = R - \Delta R$, go to step (1).
else
  $regression\_line \leftarrow least\_square(P^i_{start})$
  // $regression\_line$ has two intersection points
  // with the neighborhood circle, O_1 and O_2.
  $P_{ave} = \sum_{k=1}^{n} P_k/n$.
  if $\|P_{ave} - O_1\| > \|P_{ave} - O_2\|$ **then**
    $\hat{s}^*_i = (O_2 - O_1)/\|O_2 - O_1\|$
  else
    $\hat{s}^*_i = (O_1 - O_2)/\|O_2 - O_1\|$
  end if
end if

Step 2. Construct neighborhood circle C centered at $P_c = P^i_{start} + R\hat{s}^*_i$ with radius R.

Select all points P_k from C, such that $\|P_k - P_c\| \leq R$, to form a data set C_i.
Compute ρ pf data set C_i.

Algorithm A.2 The algorithm for finding the radius of the neighborhood of S^i.

Step 3. **if** $\rho > \rho_{high}$ **then**

$R = R + 2\Delta R$, go to step (4).

else if $\rho < \rho_{low}$ **then**

$R = R - \Delta R$, go to step (4).

else

Return P^i_{start}, P^{i*}_{end} and all points from C_i. Stop.

end if

Step 4. $regression\_line \leftarrow least\_square(P^i_{start})$

// $regression\_line$ has two intersection points

// with the neighborhood circle C, P^i_{start} and P^{i*}_{end}.

Set $\hat{s}^*_i = (P^{i*}_{end} - P^i_{start})/\|P^{i*}_{end} - P^i_{start}\|$

Repeat step (2).

Since we do not have any prior knowledge about the neighborhood of S^i, i.e. the unit oriented vector $\hat{s}_i$, we need to find a reasonable estimate to start the iterative process. This is achieved in step (1) of Algorithm A.2. We start by choosing a small R to create a neighborhood circle (centered at P^i_{start} as shown in figure A.3) such that the points within this circle have a good linearity. We then compute a regression line that passes through P^i_{start} which help determine a good estimate of $\hat{s}^*_i$. From step (2), we start with a neighborhood circle (centered at $P_c = P^i_{start} + R\hat{s}^*_i$) and adaptively find the maximal allowable neighborhood radius. The example shown in figure A.3 shows this process for the construction of S^2. From the final neighborhood circle of S^2, P^{2*}_{end} is obtained. The closest point to P^{2*}_{end} within this neighborhood, is then found and used as P^2_{end}.

The outputs from the above procedure are P^i_{start} and P^i_{end} and all the points from C_i. Using $P^i_{start}P^i_{end}$ as the diameter, a new neighborhood circle is obtained, we then delete all the other points within this circle. The remaining planar data set C is also updated. The above algorithm is then applied to construct S^{i+1}, until the remaining planar data set C is null. At the end, all points P^i_{start} and P^i_{end}, $(i = 1, .., n)$ form a feature poly-line consisting of n feature points.

A.2.1 Reason for Rejecting this Approach

This method seems to have nice results on 2D data sets like those illustrated in figures A.1 and A.3, but when it is applied to our own clouds, the results were not as expected. In the example of figure A.4, the line segment of a region shows a significant deviation from the line defined by the data points of this region. The deviation of the resulting line and the data set is caused by the non-uniform dispersion of the points of the cross section. The problem has been caused by the projection of 3D the points onto the 2D plane. While the scanning process has provided a point cloud with points evenly sparsed on the 3D space, according to the resolution of the 3D scanner, the process of projecting the slice points onto their 2D plane has brought some points very close to each other, while some regions between the points have been left empty. The number of points and their positioning affects the least-square fitting method, causing the line segment to deviate from its desired path.

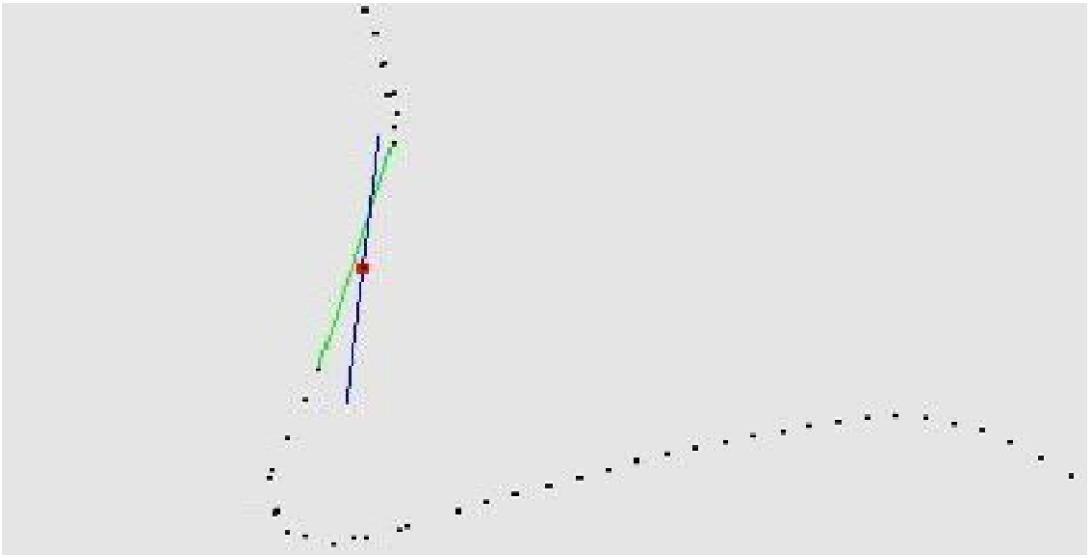


Figure A.4: When applied to parts of a cross section of the screwdriver point cloud, the thinning method using the correlation coefficient does not give satisfying results. For this reason, this approach has been rejected.

For this reason we chose to abandon the approach of using a thinning method for constructing the feature poly-line. Instead, we decided to use the computational geometry method described in chapter 4.

A.3 The Alternating Convex Hull

Another method we studied, developed, implemented, and eventually rejected, was the *alternating convex hull* method, which was applied immediately after the computation of the convex hull (section 4.3). According to this approach, when the convex hull is computed, each line segment of the convex hull defines a region, and the points of this neighborhood are assigned to their corresponding region, according to the definitions described in section 4.4.

Some of the regions are not properly described by the feature poly-line, as there are some points that belong to these regions but are not close to the curve. We treat further these remote regions by computing the convex hull of their points separately, and then we repeat the same process as for the initial object, i.e. we add the vertices of their convex hull to the closed curve and update the feature poly-line. This is illustrated in figure A.5, where we can see a cross section of the handle of the screwdriver being processed. The convex hull forms the initial feature poly-line and the points of the cross section form regions, some of which are close to the poly-line, whereas other regions consist of points that need further processing. One such region is highlighted. There are other regions that also have points positioned far from the poly-line at other parts of the cross-section. When these regions are processed one by one, the vertices of their convex hull are added to the feature poly-line, and the regions are re-computed.

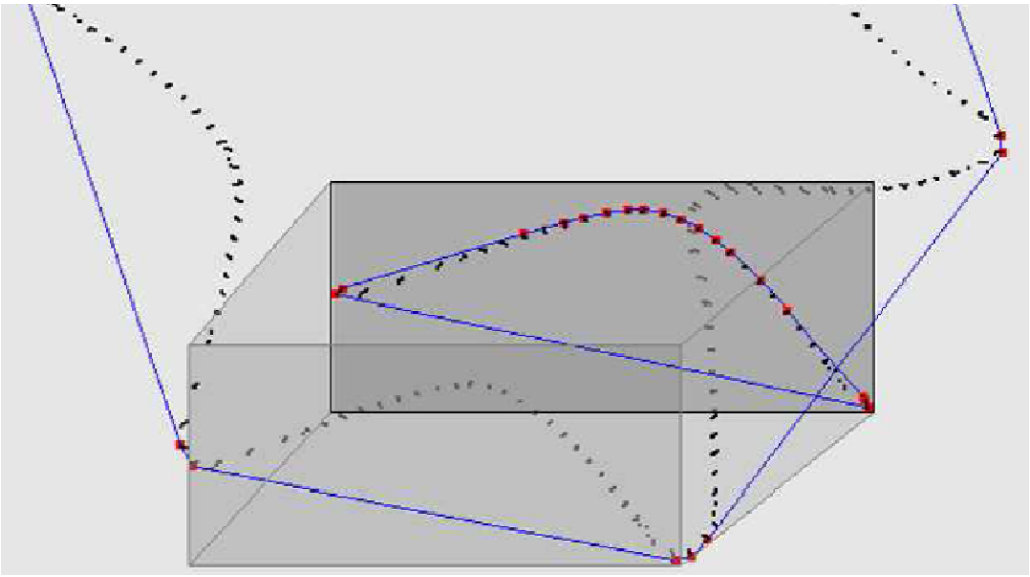


Figure A.5: Detail of a cross section of the handle of the screwdriver point cloud. The points in the highlighted region are isolated and the convex hull of these points is computed to identify the next set of feature points for the curve.

By combining the initial convex hull with the convex hull of each region, we get a curve that interpolates the points of the slice adequately in most regions. We repeat computing the convex hull for the rest of the regions until all regions consist of points that are located near the curve. As stated in algorithm A.3, the average distance is used for determining whether the points of a region are close to the curve or not, and the process is repeated as long as there have been any changes to the curve in the previous step. A similar technique is described in [76].

An example is illustrated in figure A.6. In this example, all concave regions of figure A.5 have been processed and the feature poly-line has been updated with the new points. The regions have been recomputed, and now in figure A.6(a) there are two regions that obviously need one more iteration. The second iteration is performed and the regions are updated with their new feature points, as illustrated in figure A.6(b).

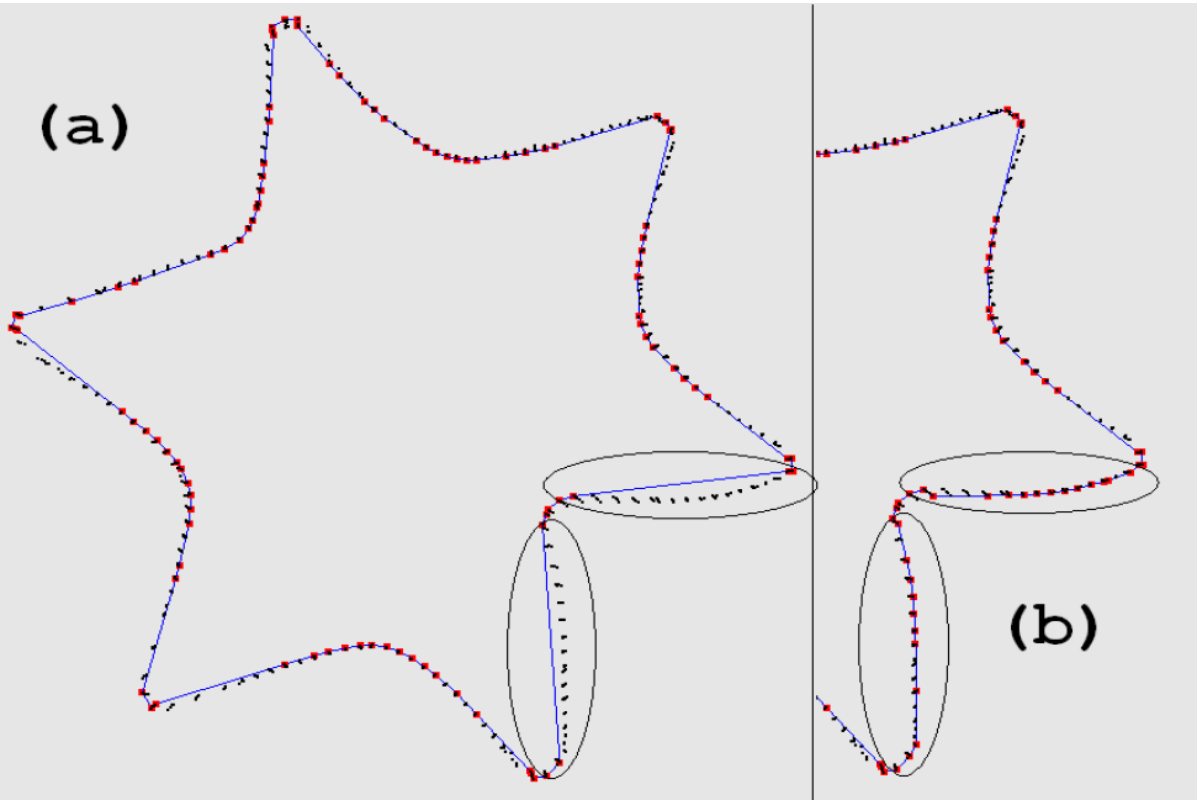


Figure A.6: A cross section of the handle of the screwdriver point cloud. (a) Second iteration of the Alternating Convex Hull Method. There are still some regions where the curve doesn't fit the points accurately. (b) Third iteration of the Alternating Convex Hull Method. The curve now describes the slice points more accurately.

The method, as applied to a specific slice, is summarized as follows:

Algorithm A.3 The algorithm for the Alternating convex hull.

alternating_convex_hull()

Input: a set P of points, Slice i

Output: an ordered set F_i of feature points

step 1: (P_i^{3D}, L) = slice(i, P)

step 2: P_i = project(P_i^{3D}, S)

step 3: F_i = qconvex(P_i)

step 4:
for each region P_{ij} of F_i **do**
if avg dist$(P_{ij}, F_{ij}) > e$ **then**
F_{ij} = qconvex(P_{ij} - F_{ij})
end if
end for
$F_i = F_i \cup F_{ij}$
if $changes\_made = true$ **then**
repeat step 4
end if

step 5: return F_i

where L is a plane parallel to the slice where we project the 3D slice.

A.3.1 The Internal - External Point switching effect

This approach has also been abandoned - after being fully implemented - for the reason that the feature poly-line presented some irregularities at the points where the initial convex hull and the regional convex hulls were linked together. The problem arises due to fact that the points of a cross section are usually arranged on a wider area rather than a mere curve. Considering that the points of the 3D cloud lie on the surface of the object, we would expect the points of a 2D slice to lie on a curve with

no thickness. But since we allow for slices of certain adaptable thickness, projecting the points on a plane produces point regions that form a thickened curve. If the dispersion of the points is insignificant, the resulting curve would be sufficient. If not, computing the convex hull on each region would result in selecting external points on some parts of the slice, and internal points on other parts (see figure A.7). We call this the *internal-external point switching effect*.

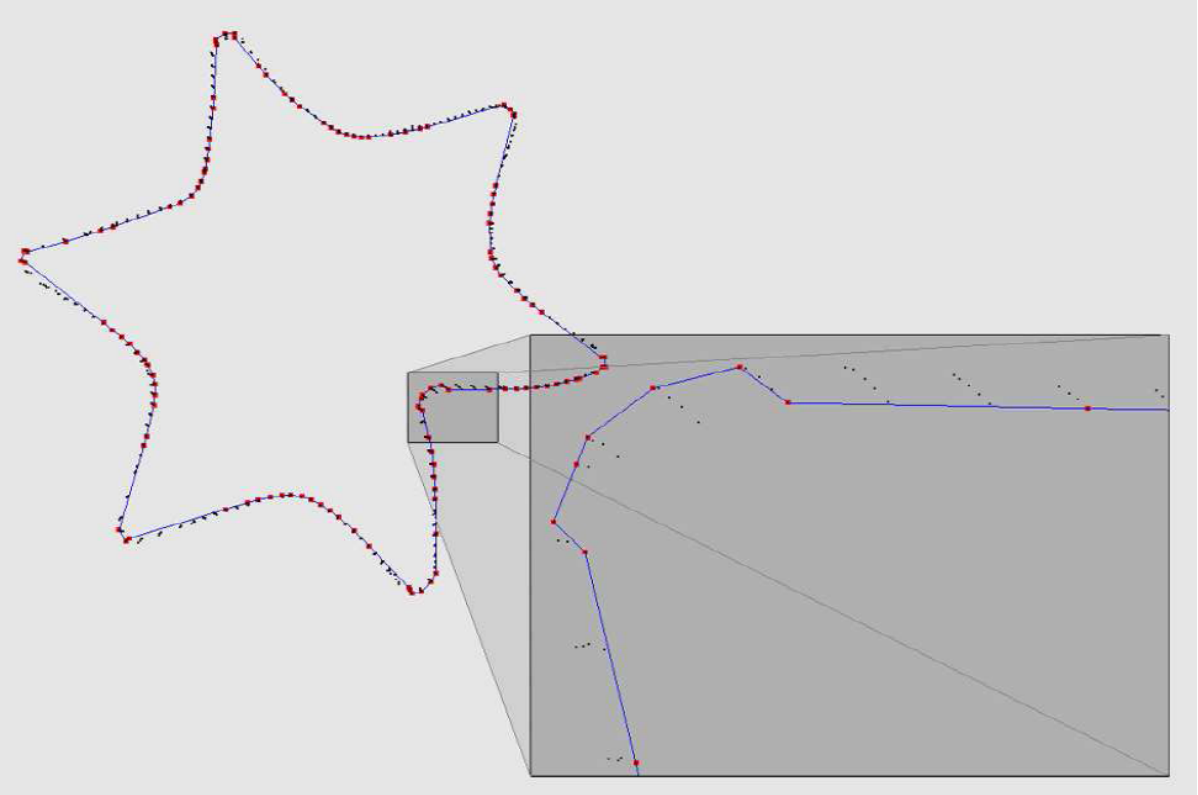


Figure A.7: A detail of a cross section of the handle of the screwdriver point cloud. The convex hull method identifies internal and external slice points alternatively as feature points. If this affects the curve significantly, the feature poly-line will have irregularities when switching between regions.

To overcome this effect we used the alternative approach described in the section 4.5, which identifies external feature points only using the Voronoi diagram of the regions.

A.4 The Voronoi diagram based approach - Variations

While developing the method of section 4.5, which uses the Voronoi diagram of a region to identify additional (external) points for the feature poly-line, we studied and implemented several variations before concluding to the algorithm 4.1. They are discussed below, with purpose to highlight the details that render the final method satisfactory, while the previous variations contain faults. In the discussion of the next section, we assume we have already constructed the initial curve with the convex hull of the cross section. We also have computed the regions, and for those regions not properly described by the curve, we have already computed their Voronoi diagram.

A.4.1 Using the farthest Voronoi Vertex

In a first attempt we may choose the farthest Voronoi vertex located on the outer side of the point cloud, and locate the three (or more) points that are on the largest empty circle for this vertex (see Figure A.8). Since we choose the farthest Voronoi vertex, two of these points are expected to be the extreme feature points of this region, which are already known from the previous step. So we expect one or more remaining points to be identified as new feature points.

This is equivalent to choosing the third point of the unique Delaunay triangle that the two already known feature points participate in, considering that there are not more than three co-circular points for this Voronoi vertex (if there are more, the triangle is not unique, but still we can choose any of the points, or even all of them). We identify these new points as feature points and update the feature poly-line accordingly. We continue to the next farthest Voronoi vertex at the new region and repeat the process, until all points are located near the fitting feature poly-line. Every time we update the feature poly-line, we re-compute the point distances from the regions. This also ensures that we do not reuse points already near the poly-line. The method is summarized in algorithm A.4.

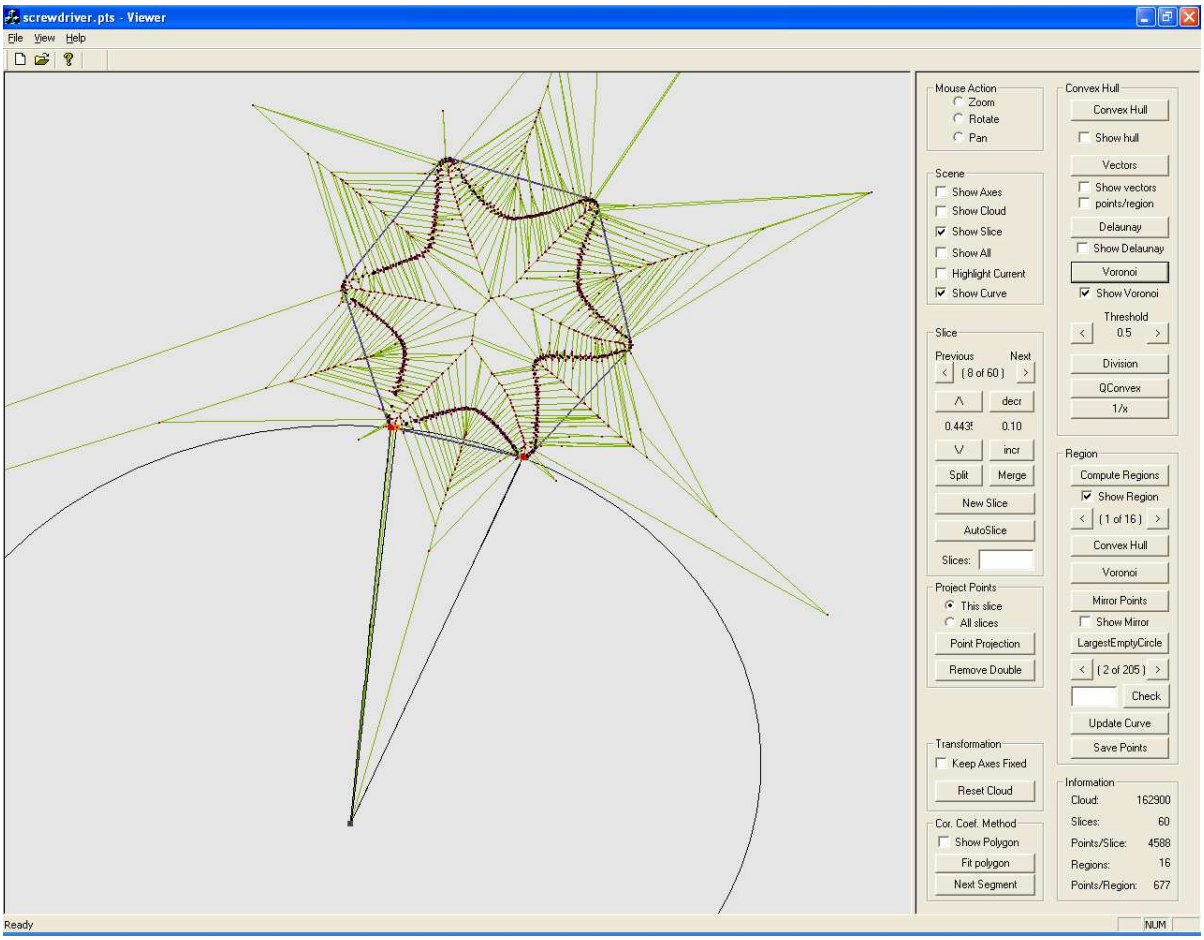


Figure A.8: A cross section of the handle of the screwdriver point cloud. The largest empty circle of the farthest Voronoi vertex is used to identify the next group of feature points.

By choosing the farthest Voronoi vertex every time, the feature points selected are expected to be relatively close to each other. If we do not restrict the Voronoi vertices it will take many steps to fully update the fitting feature poly-line for this region. The final poly-line will consist of more feature points, and will describe the point cloud slice more accurately. The problem is that the feature poly-line approaches the point cloud area quite slowly. To speed up the process we apply the variation described in the following section.

Algorithm A.4 The algorithm for the Voronoi method, using the farthest Voronoi vertex.

Voronoi_farthest_vertex()

Input: a set P of points, Slice i

Output: an ordered set F_i of feature points

step 1: $(P_i^{3D}, L) \leftarrow slice(i, P)$

step 2: $P_i \leftarrow project(P_i^{3D}, S)$

step 3: $F_i \leftarrow qconvex(P_i)$

step 4: $F_{ij} \leftarrow \emptyset$

step 5:
repeat
for each region P_{ij} of F_i **do**
if avg dist$(P_{ij}, F_{ij}) > \varepsilon$ **then**
$V_i \leftarrow qvoronoi(P_{ij})$
$V_{max} \leftarrow farthest\_vertex(V_i, P_{ij})$
$F_{ij} \leftarrow largest\_circle(V_{max}, P_{ij})$
end if
end for
$F_i \leftarrow F_i \cup F_{ij}$
until $F_{ij} \neq \emptyset$

step 6: return F_i

A.4.2 Using a single Voronoi Vertex from a bounded area

What we do here is to restrict the candidate Voronoi vertices within a bounded area. With this restriction, the process will take fewer steps to update the feature poly-line. It is usually safe to ignore some Voronoi vertices, which are located too far from the region, and choose a Voronoi vertex that is the farthest within a bounded area. In the example of figure A.9 the bounded area is defined as the mirror of the convex hull of the region being processed. All Voronoi vertices located outside this region

are ignored during the calculations.

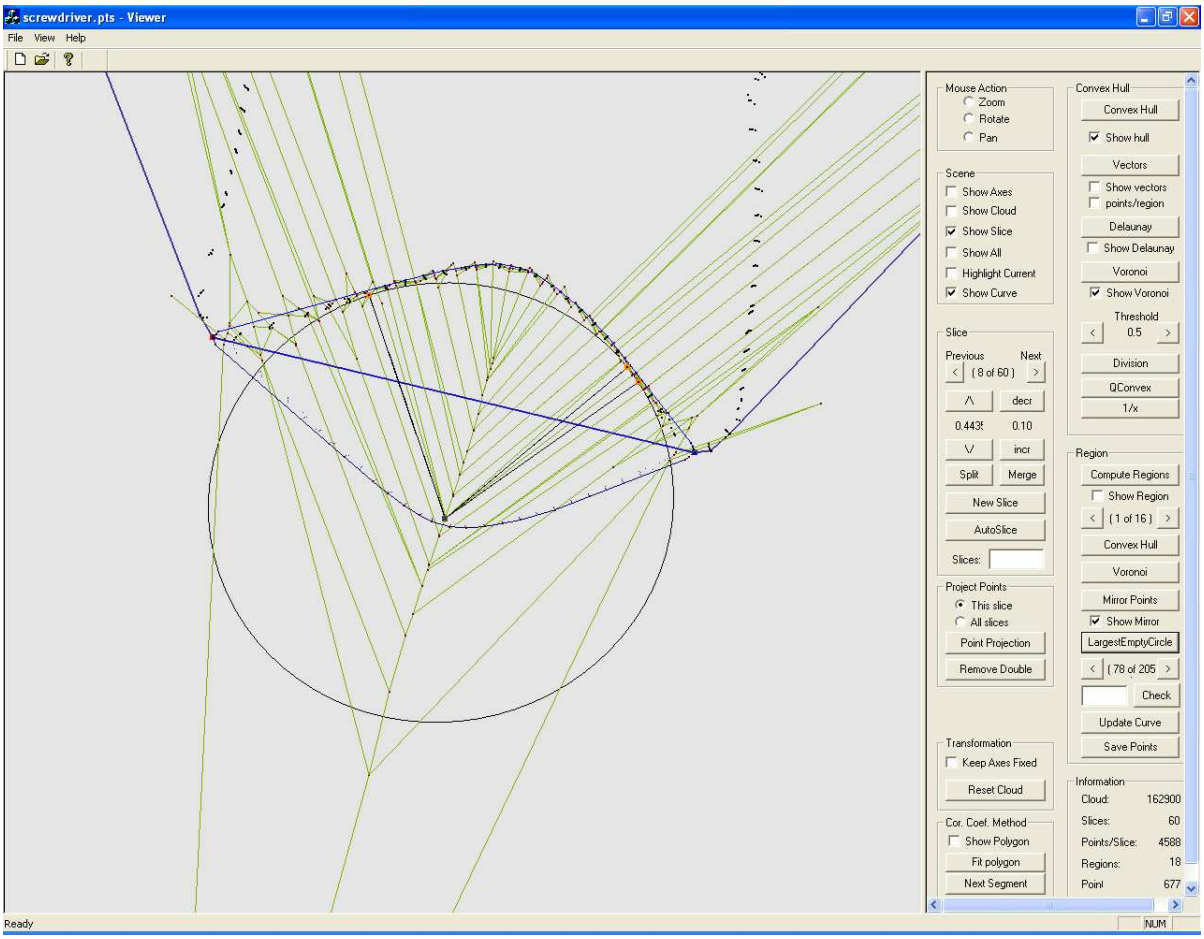


Figure A.9: A cross section of the handle of the screwdriver point cloud. By choosing a Voronoi vertex that is the farthest inside a bounded region we can identify feature points that are not too close to each other, and update the fitting poly-line faster.

Using this variation, the feature points will be fewer, with a possible compromise in terms of fitting accuracy. We use this parameter to adapt the feature poly-line fitting according to application specific requirement or quality of approximation guaranties. The algorithm for this variation is as follows.

Algorithm A.5 The algorithm for the Voronoi method, using the farthest Voronoi vertex within a bounded area.

Voronoi_farthest_vertex_bounded_area()

Input: a set P of points, Slice i

Output: an ordered set F_i of feature points

step 1: $(P_i^{3D}, L) \leftarrow slice(i, P)$
step 2: $P_i \leftarrow project(P_i^{3D}, S)$
step 3: $F_i \leftarrow qconvex(P_i)$
step 4: $F_{ij} \leftarrow \emptyset$
step 5:
repeat
for each region P_{ij} of F_i **do**
if avg dist$(P_{ij}, F_{ij}) > \varepsilon$ **then**
$V_i \leftarrow qvoronoi(P_{ij})$
while V_i is not inside bounded area **do**
$V_{max} \leftarrow farthest\_vertex(V_i, P_{ij})$
$remove\_V_{max}\_from\_V_i$
end while
$F_{ij} \leftarrow largest\_circle(V_{max}, P_{ij})$
end if
end for
$F_i \leftarrow F_i \cup F_{ij}$
until $F_{ij} \neq \emptyset$
step 6: return F_i

The only difference from the algorithm A.4 is that whenever we look for the farthest Voronoi vertex V_{max}, we ignore the Voronoi vertices that are located outside the bounded area.

With this variation we managed to speed up our method significantly. But during our tests, we found another way to make things go faster.

A.4.3 Using multiple Voronoi Vertices

Another variation that speeds up the method significantly is to choose more than one Voronoi vertices in each step, and process many slice points simultaneously. Instead

of choosing one Voronoi vertex and locate the contact points of the largest empty circle for this vertex, we select many Voronoi vertices and locate all the contact points for all such largest empty circles. Then we update the feature poly-line with all the point cloud feature points we have located. An example is illustrated in figure figure A.10.

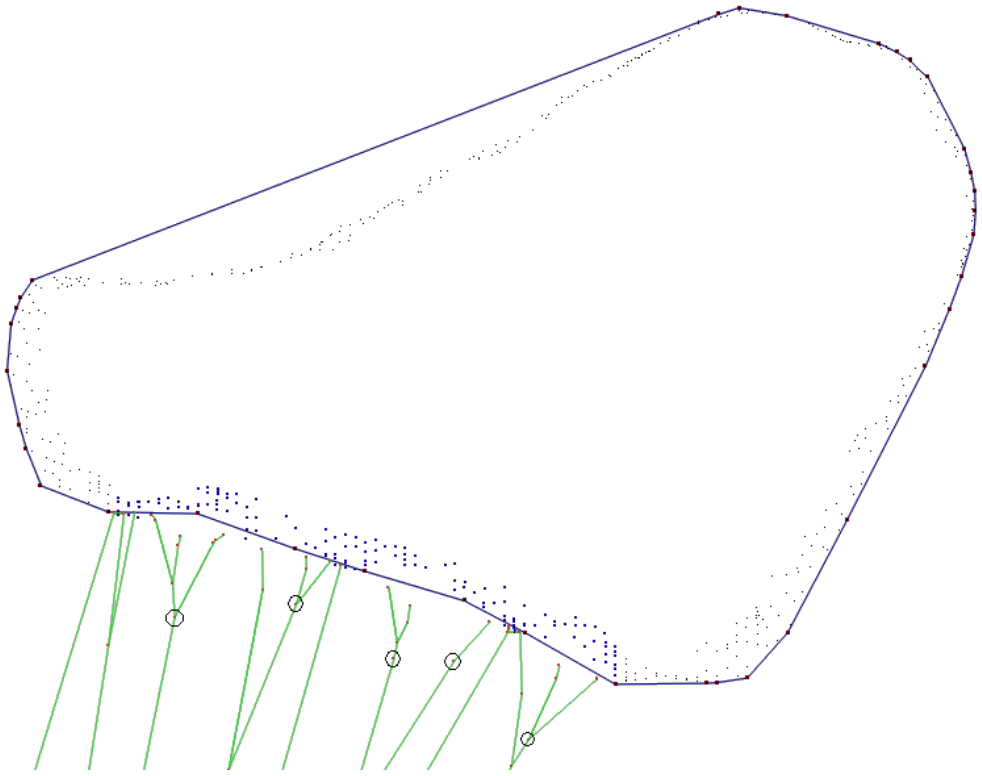


Figure A.10: A cross section of the hip bone point cloud. By choosing several Voronoi vertices at once we can update the fitting poly-line even faster.

This alternative differs from the methods described previously as we perform the work of many steps in just one step, and the feature poly-line is extracted much faster. Voronoi vertices that are close to the point cloud do not contribute any information to the feature point set so they are being left out. Furthermore, Voronoi vertices on the wrong side of the point cloud are also being excluded.

The algorithm for this variation is the same as described in algorithm A.5, but this time V_{max} is not a single vertex, but a set of Voronoi vertices. This is actually the algorithm described in section 4.5, which is the final approach we used in our method. In algorithm 4.1 the set of vertices $V_{candidate} \leftarrow V_i - excluded\_Voronoi\_vertices$ is equivalent to the **while** statement (step 5 in algorithm A.5).

A.5 The Rational Bezier Curve Fitting Approach

Before adopting and implementing the method described in section 5.3 for representing the cross-sectional contours by G^1 B-Spline curves, we tried another approach, which used rational Bezier curves for interpolating the feature poly-line. According to this method, which was proposed by Stamati and Fudos in [14], the purpose is to find the curve that best approximates the feature poly-line set to represent the general morphology of the border. An optimization process is used, in which cubic rational Bezier curves are fit to the feature boundaries, while ensuring that the curves created conform to the conditions required for G^1 continuity. They used an equivalent instance of the general NURB, namely piecewise rational cubic Bezier curves because the constraints they applied decrease the degrees of freedom of our problem and our requirements are well met with this low degree simpler representation resulting in fast converging optimization algorithm. Using piecewise rational Bezier curves they basically follow an optimization approach which can inherently rule out noisy data without affecting the shape of the boundary as a whole.

Curve approximation is carried out with a least squares optimization procedure. Suppose $Q = \{Q_1, Q_2, ...Q_m\}$ is a set of ordered border points and C is an approximating rational Bezier curve given by the equation

$$C(u_i) = \frac{\sum_{j=1}^{n} w_j P_j B_j(u)}{\sum_{j=1}^{n} w_j B_j(u)} \qquad \text{(eq 1.4)}$$

where $n = 4$ for a cubic rational Bezier curve, u_i is the parameter value associated with border point Q_i, P_j are the control points, w_j is the weight of each control point and B_j is the corresponding Bernstein polynomial.

Assuming that all points of Q should be approximated by the curve, we would like to minimize the error:

$$e_i = Q_i - C(u_i), i = 1..m. \qquad \text{(eq 1.5)}$$

We need to assign parameter values u_i to each point Q_i. We use chordal parameterization [181] in which we express the parameter value of each point Q_i in reference to its position in the point sequence:

$$u_i = \frac{\sum_{j=2}^{i} \Delta Q_j}{\sum_{j=2}^{m} \Delta Q_j} \qquad \text{(eq 1.6)}$$

The least squares problem is then to minimize the error:

$$E = \sum_{i=1}^{m} {\epsilon_i}^2 \qquad \text{(eq 1.7)}$$

which by its turn, when combined with equations eq 1.4 and eq 1.5 becomes:

$$E = \sum_{i=1}^{m} (Q_i - C(u_i))^2 \qquad \text{(eq 1.8)}$$

We consider the product $w_j P_j$ as one variable and partially differentiate equation eq 1.8 by factor $w_k P_k, k = 1..4$. This leads to equations:

$$\frac{\partial E}{\partial w_k P_k} = 0 \Rightarrow \sum_{i=1}^{m} \Big[2(\sum_{j=1}^{4} w_j P_j B_j(u_i) - Q_i \sum_{j=1}^{4} w_j B_j(u_i)) B_k(u_i) \Big] = 0, k = 1..4 \qquad \text{(eq 1.9)}$$

from which we obtain the following linear system of equations:

$$\begin{aligned} [B]^T[B][wP_x] &= [B]^T[Q_x B][w] \\ [B]^T[B][wP_y] &= [B]^T[Q_y B][w] \\ [B]^T[B][wP_z] &= [B]^T[Q_z B][w] \end{aligned} \qquad \text{(eq 1.10)}$$

where (i.e. for coordinate x):

$$B = \begin{bmatrix} B_1(u_1) & B_2(u_1) & B_3(u_1) & B_4(u_1) \\ \vdots & & & \vdots \\ B_1(u_1) & B_2(u_1) & B_3(u_1) & B_4(u_1) \end{bmatrix}, wP_x = \begin{bmatrix} w_1 P_{x_1} \\ w_2 P_{x_2} \\ w_3 P_{x_3} \\ w_4 P_{x_4} \end{bmatrix},$$

$$Q_x B = \begin{bmatrix} Q_1 B_1(u_1) & Q_1 B_2(u_1) & Q_1 B_3(u_1) & Q_1 B_4(u_1) \\ Q_2 B_1(u_1) & Q_2 B_2(u_1) & Q_2 B_3(u_1) & Q_2 B_4(u_1) \\ \vdots & & & \vdots \\ Q_m B_1(u_1) & Q_m B_2(u_1) & Q_m B_3(u_1) & Q_m B_4(u_1) \end{bmatrix}$$

Without affecting the shape of the curve or the parametrization we can assume that one of the weights is 1. Therefore we assume that weight $w_2 = 1$. We could eliminate one more weight by re-parametrizing u, but we need re-parametrization to be a parameter in the optimization process. Also to ensure G^1 continuity between Bezier curves we make sure the starting point of one curve coincides with the end

point of the previous curve and that the inner control points are located accordingly on the tangents of the end points.

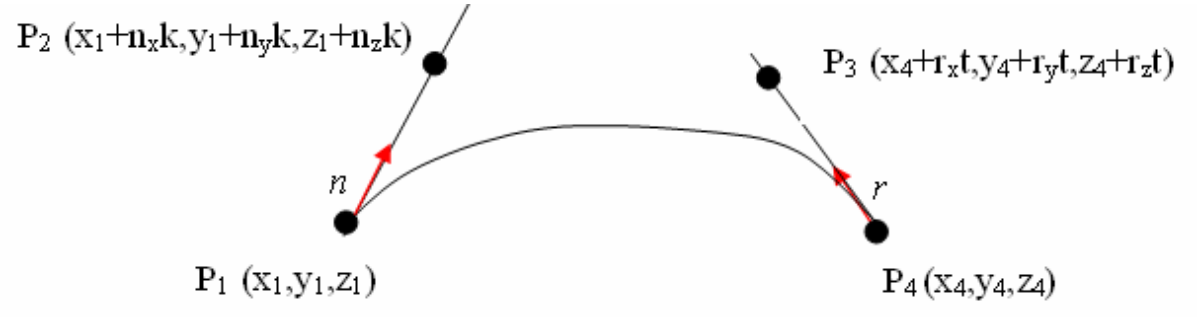


Figure A.11: The unit vectors of the tangents at the end control points are estimated by expressing each tangent of each point as a linear combination of its 4 closest neighbors [3].

Vectors wP_x, wP_y, wP_z are modified by expressing the coordinates of inner control points P_2 and P_3 in relation to the end points. In equation eq 1.10 we now have:

$$[B^T][B]\begin{pmatrix} w_1(x_1+0) \\ w_2(x_1+n_xk) \\ w_3(x_4+r_xt) \\ w_4(x_4+0) \end{pmatrix} = [B^T][Q_xB][w] \qquad \text{(eq 1.11)}$$

Since we assume that $w_2 = 1$, the system is transformed so that its final form is (e.g. for direction x):

$$[B^T][B]\begin{pmatrix} w_1x_1 \\ n_xk \\ w_3r_xt \\ w_4x_4 \end{pmatrix} = [B^T][Q_xB][w] - [B^T][B][A_x\, where [A_x] = \begin{bmatrix} 0 \\ x_1 \\ w_3x_4 \\ 0 \end{bmatrix} \qquad \text{(eq 1.12)}$$

These systems of linear equation (for directions x, y and z) can be used to derive values for variables w_1, k, t, and w_4 (all weights in vector $[w]$ are initialized to 1), therefore essentially determining the inner control points coordinates and approximate values for two of the three weights. To achieve better curve approximation, we proceed to a second step using the control points calculated above to compute more appropriate weight values. However, for each system solution we obtain a different

set of variable values. Therefore the following weight optimization procedure is carried out once for every solution set and we accordingly keep the solution that best minimizes the least squares error.

Specifically, we express equation eq 1.7 as follows:

$$\epsilon_i = \sum_{i=1}^{m} (\epsilon_{x_i}^2 + \epsilon_{y_i}^2 + \epsilon_{z_i}^2) \qquad \text{(eq 1.13)}$$

We partially differentiate by weights w_k $(k = 1, 3, 4)$ and obtain a system of equations from which we can substitute the control points and the weights found in the previous step and optimize the weight vector. Specifically, for $\partial E/\partial w_k = 0, k = 1, 3, 4$, the equations obtained are of the form:

$$\sum_{i=1}^{m} \left(B_k(u_i) \left(Q_{x_i} Q_{x_i} \sum_{j=1}^{4} w_j B_j(u_i) + Q_{y_i} Q_{y_i} \sum_{j=1}^{4} w_j B_j(u_i) + Q_{z_i} Q_{z_i} \sum_{j=1}^{4} w_j B_j(u_i) \right) \right) =$$
$$\sum_{i=1}^{m} \left(B_k(u_i) \left(Q_{x_i} \sum_{j=1}^{4} w_j P_{x_i} B_j(u_i) + Q_{y_i} \sum_{j=1}^{4} w_j P_{y_i} B_j(u_i) + Q_{z_i} \sum_{j=1}^{4} w_j P_{z_i} B_j(u_i) \right) \right) \qquad \text{(eq 1.14)}$$

Eventually we end up with the linear system:

$$\begin{aligned} &\big([Q_x^{'}B]^T[Q_x^{'}B] + [Q_y^{'}B]^T[Q_y^{'}B] + [Q_z^{'}B]^T[Q_z^{'}B]\big)[w^{'}] = \\ &[Q_x^{'}B]^T[B][w^{'}P_x^{'}] + [Q_y^{'}B]^T[B][w^{'}P_y^{'}] + [Q_z^{'}B]^T[B][w^{'}P_z^{'}] + \\ &(-1)\big([Q_x^{'}B]^T[C_x] + [Q_y^{'}B]^T[C_y] + [Q_z^{'}B]^T[C_z]\big) \end{aligned} \qquad \text{(eq 1.15)}$$

where (e.g. for x coordinate):

$$Q_x^{'}B = \begin{bmatrix} Q_1B_1(u_1) & \cdots & Q_1B_4(u_1) \\ \vdots & \ddots & \vdots \\ Q_mB_1(u_m) & \cdots & Q_mB_4(u_m) \end{bmatrix} \; and \;\; C_x = [Q_x^{'}B] \begin{bmatrix} P_{X_2}B_2(u_i) \\ \vdots \\ P_{X_2}B_2(u_m) \end{bmatrix}$$

This procedure is carried out iteratively until the error function is minimized.

Generally this is a fast curve approximation approach that produces smooth continuous curves that interpolate or pass close by the data points. A good approximation is reached within a few iterations. In some cases, the minimal error is reached after the first iteration, if the point cloud data is not noisy and the tangent estimations are

good. The accuracy of the approximating curve basically depends on how well the tangents are estimated at the end points, since we restrict the inner control points to be located on them. The weights are used not only to determine the shape of the curve but also to adjust the parameterization of the curve. Chordal parameterization works well, assuming that the points are given as a sequence in 3D space. For this reason we transform all points that we want to approximate by translating P_1 to the origin, and then aligning vector $(P_4 - P_1)$ to the positive $z - axis$. Then we simply sort the points Q_i according to their z coordinate by:

$$Q_i^{'} = A(\mathrm{v})T(-P_{x_1}, -P_{y_1}, -P_{z_1})Q_i \qquad \text{(eq 1.16)}$$

where

$$\mathrm{v} = [P_4 - P_1] = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, A = \begin{bmatrix} \frac{l}{|\vec{\mathrm{v}}|} & \frac{-ab}{l|\vec{\mathrm{v}}|} & \frac{-ac}{l|\vec{\mathrm{v}}|} & 0 \\ 0 & \frac{c}{l} & \frac{-b}{l} & 0 \\ \frac{a}{|\vec{\mathrm{v}}|} & \frac{b}{|\vec{\mathrm{v}}|} & \frac{c}{|\vec{\mathrm{v}}|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, and \quad l = \sqrt{b^2 + c^2}$$

This curve fitting method was implemented using Maple 11 [182]. The data sets used in the examples are ordered point sets corresponding to the feature points of our cross sections, i.e. the feature poly-lines. For each point in the border point cloud we have previously computed its corresponding surface normal vector estimate, according to [3]. The boundary point set is divided into subsets (curve segments) based on the progressive change in the surface normal of the points. The point cloud is divided into as many sets needed, so that the angle formed by the surface normals of the start and end point of each section is below a threshold and allows for alignment of the section's point data and sorting as described in equation eq 1.16.

The following figures illustrate details of the procedure, when applied to a cross section of the cycladean idol model. Figure A.12 shows the tangent vectors of the feature points as evaluated using the method described in [3]. This method calculates an estimation of the normal vector using the neighborhood of the point in discussion. The tangent vector is then easily calculated, as it is vertical to the normal vector.

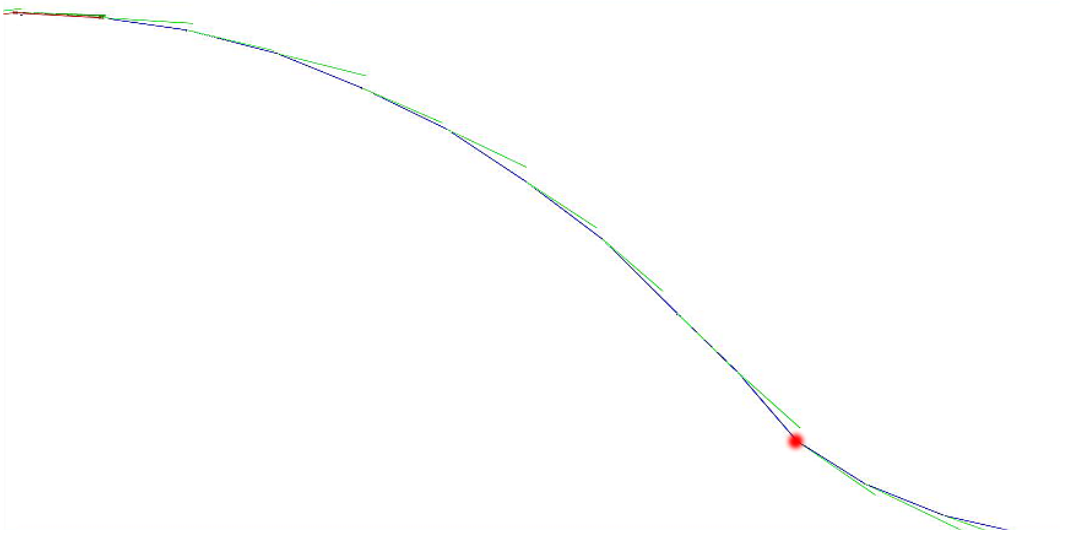


Figure A.12: A new rational Bezier curve is inserted each time the tangent vectors switch sides in respect to the direction of the feature poly-line.

Using this criterion, we divide the feature poly-line in a number of segments, which will be represented as individual rational Bezier curves. Of course these curves have to be smooth and continuous at their joints, so we set the corresponding control points to be collinear. In other words, the control point where two curves meet, the previous control point on the one curve, and the next point on the other curve, have to be collinear, to ensure a smooth continuous transition from one curve to another. Figure A.13 shows how the feature poly-line is segmented into regions for fitting the individual rational Bezier curves.

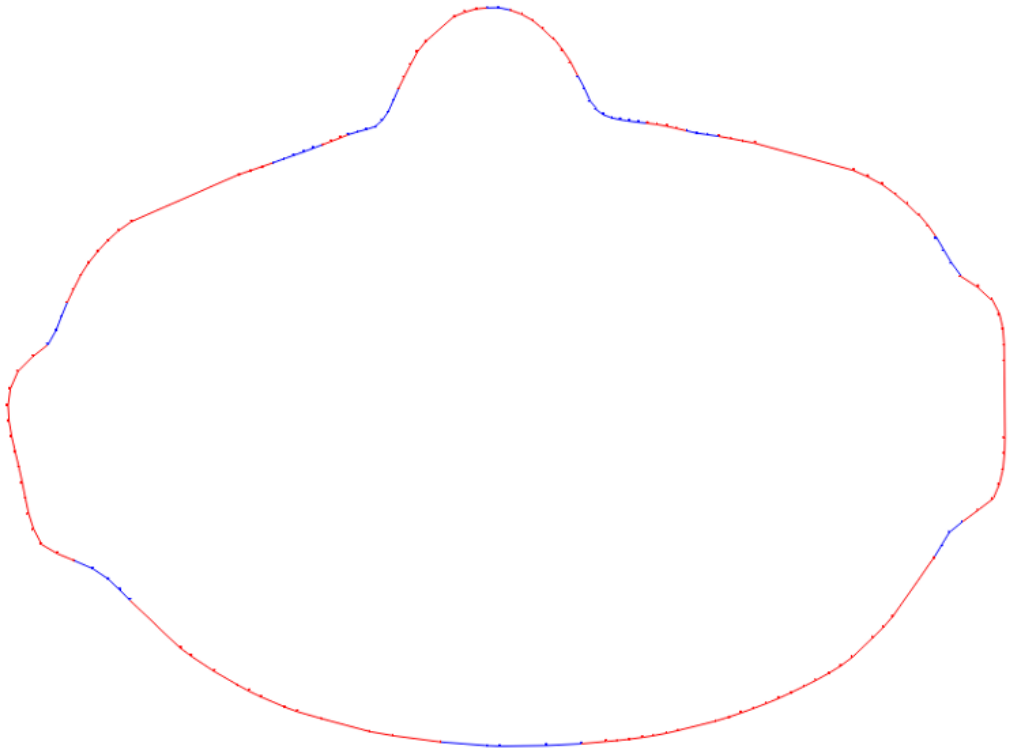


Figure A.13: The feature poly-line is segmented into regions for fitting the individual rational Bezier curves. Continuity and smoothness on the edges is ensured by setting the control points on the edges to be collinear.

A.5.1 Reason for Rejecting this Approach

The process described above should work well, but the results when applied to our models where rather disappointing. As illustrated in figure A.14, the over-fitting effect we discussed in section 5.3 is present in many parts of the model. The reason for this is that our input point list, i.e. the feature poly-line of a cross section, as a result of the computations, may contain points that are positioned very close to each other. The fitting curves are forced to interpolate all points in the feature poly-line, and consequently they fail to describe the model accurately due to over fitting. In their method, Stamati and Fudos [14] have used as input a ”thinned out” point set, by representing groups of neighboring points with the border point that is closest to their center of mass. For such point sets, the method obviously produces accurate results.

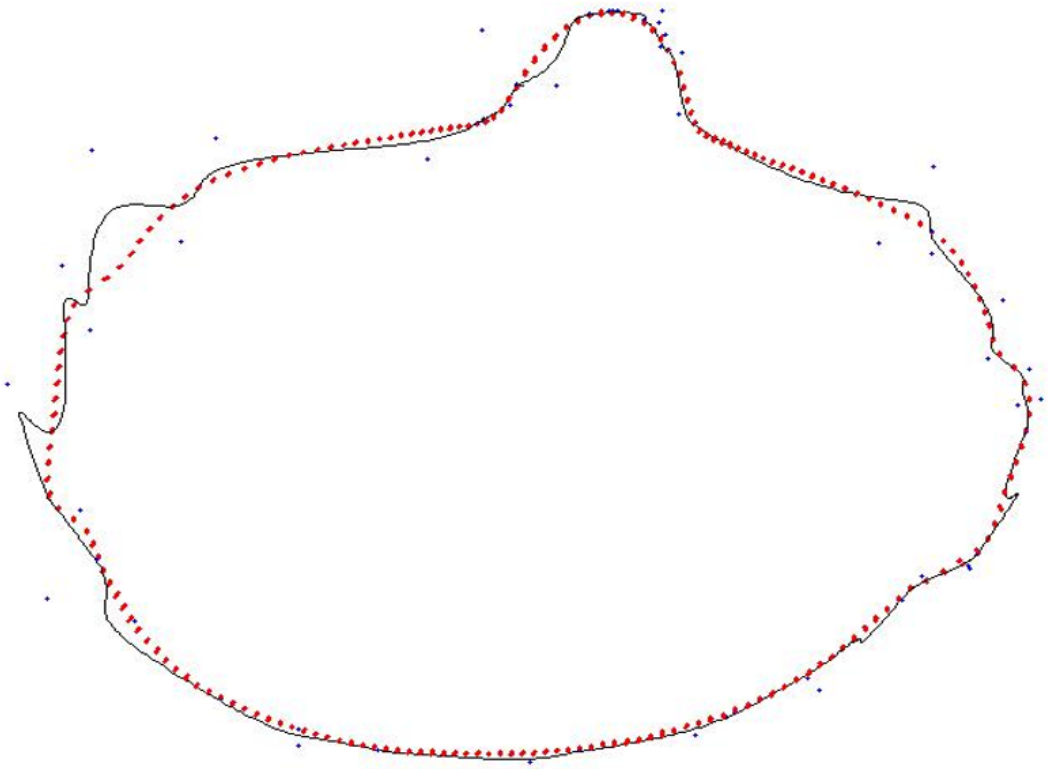


Figure A.14: The over fitting effect is present in many parts of the model. Removing feature points that are close to each other will solve the problem. However, the number of rational Bezier fitting curves that have to be calculated again is large.

The problem of over-fitting may be present regardless of the type of curve we choose to interpolate on our point set. But when it comes to removing some points (which are close to each other), this means we have to recalculate the curves, trying to achieve smooth continuous results that describe the model accurately. With the approach described here we would need to recalculate many curves, increasing the computations significantly.

The solution we chose was to calculate a single B-Spline and not many rational Bezier curves. In case a curve has to be recalculated, only one curve has to be constructed for each cross section, and for this reason we chose the approach described in section 5.3 which uses one B-Spline curve instead of rational Bezier curves.

Author's Publications

I. Fudos and I. Kyriazis. **Thin Client Access to a Visualization Environment**. 3rd International Workshop on Computer Graphics and Geometric Modelling CGGM04 held in conjunction with ICCS2004
https://link.springer.com/chapter/10.1007/978-3-540-25944-2_33

I. Kyriazis, I. Fudos and L. Palios. **Detecting Features from Sliced Point Clouds**. 2nd International Conference on Computer Graphics Theory and Applications GRAPP07
http://grapp.visigrapp.org/GRAPP2007/index.htm

I. Kyriazis, I. Fudos and L. Palios. **Extracting CAD Features from Point Cloud Cross Sections**. 17-th International Conference on Computer Graphics, Visualization and Computer Vision'2009 WSCG09
http://wscg.zcu.cz/wscg2009/wscg2009.htm

I. Kyriazis and I. Fudos. **Building Editable Free-form Models from Unstructured Point Clouds**. Computer-Aided Design and Applications, 10(6), 877-888, 2013
http://www.cadanda.com/TOCV10No6.html

I. Kyriazis and I. Fudos. **Reconstructing editable boundary representations from 3d medical data**. 2nd International Conference on Bio-Medical Instrumentation and related Engineering and Physical Sciences, Athens, Greece, 2013
http://biomep.teiath.gr/2013/index.html

I. Kyriazis and I. Fudos. **Editing Operators for Cross-Sectional Data Sets**. Computer Aided Design, Submitted, Publication pending
http://www.journals.elsevier.com/computer-aided-design

Short Biography

Ioannis Kyriazis is a PhD candidate in the Department of Computer Science and Engineering at the University of Ioannina, Greece. He received his MS degree in 2002 and his BS degree in 2000 from the University of Ioannina, both in Computer Science.

He is a member of the Distributed Management of Data (DMOD) Lab and the Computer Graphics Research Group (CGRG). His research interests include computer graphics, CAD, solid modeling and reverse engineering.

He is also an ICT teacher of Primary Education. As an ICT teacher, he is regularly participating in Education Conferences, to extend his teaching skills and update his teaching and scientific knowledge. He has also participated in training programs for ICT Teachers.