ΑΝΑΚΑΤΑΣΚΕΥΗ ΜΟΝΤΕΛΩΝ CAD ΜΕ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΒΑΣΙΣΜΕΝΗ ΣΤΗ ΜΟΡΦΟΛΟΓΙΑ ΤΟΥ ΝΕΦΟΥΣ ΣΗΜΕΙΩΝ

Η
ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από την

Βασιλική Σταμάτη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ

Οκτώβριος 2008

DEDICATION

To my husband, Thanasi

To my parents, Costa and Eugenia,
and my sister, Tonia

ACKNOWLEDGMENTS

I would like to express my sincere thanks and gratitude to my advisor 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. Theohari and Prof. Sapidi for their suggestions and insight. Also, many thanks to Prof. Trahania, Prof. Azariadi, Prof. Dimakopoulo and Prof. Lagari for their suggestions and remarks. I would also like to thank all my friends and colleagues who over the course of this research have provided me with helpful feedback.

I would also like to thank my husband, Thanasi, and my family for always being so supportive and for believing in me. I could not have done this without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ΕΚΤΕΝΗΣ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Βασιλική Σταμάτη του Κωνσταντίνου και της Ευγενίας. PhD, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Οκτώβριος 2008.
Τίτλος Διατριβής: Ανακατασκευή μοντέλων CAD με χαρακτηριστικά βασισμένη στη μορφολογία του νέφους σημείων
Επιβλέποντας: Ιωάννης Φούντος.

Η ανάστροφη μηχανική (reverse engineering) είναι μια διαδικασία μέσω της οποίας ανακατασκευάζουμε ένα γεωμετρικό μοντέλο με βάση τον υπολογιστή (CAD) από μετρήσεις που λαμβάνονται από την επιφάνεια ενός αντικειμένου, μέσω τεχνικών σάρωσης, και έχουν τη μορφή ενός νέφους σημείων. Αυτή η διαδικασία χρησιμοποιείται ευρέως σε διάφορες εφαρμογές, όπως στη παραγωγή, βιομηχανική σχεδίαση και στη σχεδίαση και αναπαραγωγή κοσμημάτων. Μπορούν να χρησιμοποιηθούν διάφορες μορφές αναπαράστασης για τα ανακατασκευασμένα μοντέλα, όπως πολυγωνικά μοντέλα, μοντέλα CSG και μοντέλα υποδιαίρεσης χώρου σε στοιχειώδη στερεά στοιχεία (voxel). Παραδοσιακά στις περισσότερες εφαρμογές χρησιμοποιούνται μοντέλα αναπαράστασης ορίων (boundary representations). Ωστόσο η σύγχρονη τάση, ειδικά στις εφαρμογές ανακατασκευής και αναπαραγωγής αντικειμένων ελεύθερης σχεδίασης, είναι η χρήση αναπαράστασης βασισμένη σε χαρακτηριστικά και περιορισμούς. Αυτό συμβαίνει διότι αυτό το μοντέλο αναπαράστασης είναι ευέλικτο και ακριβές, επιτρέπει την τροποποίηση ή/και επανασχεδιασμό του αρχικού αντικειμένου και μπορεί να εκφράσει την σχεδιαστική πρόθεση του χρήστη-σχεδιαστή.

Στην εργασία αυτή παρουσιάζεται ένα σχήμα για την ανακατασκευή αντικειμένου από νέφος σημείων με σκοπό την απόκτηση μιας παραμετρικής και επεξεργάσιμης αναπαράστασης CAD με βάση χαρακτηριστικά και περιορισμούς (feature-based and constraint-based) που να μπορεί να τροποποιηθεί και αναπαραχθεί. Σκοπός μας είναι να εντοπίζονται και να εξάγονται χαρακτηριστικά του αντικειμένου από το νέφος σημείων ώστε να είναι δυνατή η τροποποίηση ή ο επανασχεδιασμός του αρχικού

αντικειμένου, σύμφωνα με τις προτιμήσεις του χρήστη-σχεδιαστή. Επίσης, δίδεται η δυνατότητα στον χρήστη να μπορεί να εισάγει και να χρησιμοποιεί τα χαρακτηριστικά στη σχεδίαση άλλου αντικειμένου.

Πιο συγκεκριμένα, στην εργασία αυτή παρουσιάζεται μια μέθοδος για τον εντοπισμό και την εξαγωγή χαρακτηριστικών από ένα νέφος σημείων. Το νέφος σημείων πρέπει να χωριστεί σε υποσύνολα τέτοια ώστε κάθε υποσύνολο να μπορεί να αντιστοιχηθεί σε κάποιο χαρακτηριστικό (feature) του αρχικού αντικειμένου. Ορίζουμε ένα χαρακτηριστικό μέγεθος για κάθε σημείο που ονομάζουμε «point concavity intensity», το οποίο περιγράφει την ένταση κοιλότητας που εμφανίζει το αντικείμενο στο συγκεκριμένο σημείο και ορίζεται ως η μικρότερη απόσταση του σημείου από την κοντινότερη έδρα του convex hull. Αυτό το μέγεθος ουσιαστικά εντοπίζει τα χαρακτηριστικά του νέφους που βρίσκονται ανάμεσα ή που σχηματίζονται από κοιλότητες. Αναπτύξαμε έναν γρήγορο και αποδοτικό αλγόριθμο για τον υπολογισμό αυτού του χαρακτηριστικού. Επίσης αναπτύχθηκε ένας αλγόριθμος για region growing που υλοποιεί το διαχωρισμό του νέφους σημείων σε υποσύνολα που αντιστοιχούν σε χαρακτηριστικά με βάση τις τιμές του point concavity intensity κάθε σημείου και την μεταβολή του κάθετου διανύσματος της τοπικής επιφάνειας που σχηματίζουν γειτονικά σημεία. Αυτή η μεθοδολογία οδηγεί στην εξαγωγή υποσυνόλων σημείων που αντιστοιχούν σε ανεξάρτητα χαρακτηριστικά και σε υποσύνολα σημείων που αποτελούν τα όρια των χαρακτηριστικών αυτών. Κατόπιν αναπτύχθηκε ένας ικανοποιητικός, γρήγορος αλγόριθμος για την προσέγγιση των ορίων από έναν αριθμό από κυβικές ρητές καμπύλες Bezier, που προσεγγίζει τα όρια των χαρακτηριστικών με ομαλές καμπύλες που ικανοποιούν τις συνθήκες της γεωμετρικής συνέχειας G^1. Στη συνέχεια προσαρμόζονται επιφάνειες σε κάθε υποσύνολο σημείων που αντιστοιχεί σε χαρακτηριστικό, παίρνοντας τομές του υποσυνόλου και προσεγγίζοντας τα σημεία που βρίσκονται πάνω ή πολύ κοντά στην τομή με τον αλγόριθμο μας για προσέγγιση καμπυλών με κυβικές ρητές Bezier καμπύλες. Χρησιμοποιούμε τις καμπύλες που λαμβάνουμε με αυτό τον τρόπο ως οδηγούς για να «καλύψουμε» τις καμπύλες και τα σημεία με επιφάνειες. Τελικά λαμβάνουμε αυτόνομα χαρακτηριστικά σε 3Δ μορφή, στα οποία θέτουμε περιορισμούς που αφορούν, για παράδειγμα, το μέγεθος ή τον τρόπο σύνδεσης με άλλα χαρακτηριστικά. Επιπρόσθετα χρησιμοποιούμε συμμετρίες για να ανιχνεύσουμε την σκοπιμότητα χρήσης άλλων τρόπων 3Δ ανακατασκευής όπως

sweep, blending κα. Τέλος παρουσιάζουμε δύο εφαρμογές της μεθόδου μας στην ανακατασκευή κοσμημάτων και στο ταίριασμα χαρακτηριστικών σε 3Δ μοντέλα.

ABSTRACT

Stamati Vasiliki, PhD, Computer Science Department, University of Ioannina, Greece. October 2008.
Reconstructing feature-based CAD models based on point cloud morphology
Thesis Supervisor: Ioannis Fudos.

Reverse engineering, the process of obtaining a geometric CAD model from measurements obtained by scanning an existing physical model, is widely used in numerous applications, such as manufacturing, industrial design and jewellery design. In this work we propose a framework for reverse engineering objects of mechanical or freeform design to obtain fully editable feature-based CAD model that can be reproduced or modified before production. We focus on the process of detecting features on a point cloud and we present efficient methods for analyzing the morphology of the surface defined by the point cloud. We compute a point wise characteristic called point concavity intensity and we use this quantity along with the variations in the surface normal to detect regions corresponding to object features. The boundaries of the regions representing features are extracted and approximated by a collection of piecewise cubic rational Bezier curves that best fit the detected border point cloud and are G^1 continuous. We present a fast and efficient linear curve approximation approach to fit feature boundaries with such curves that can be used along with the representative feature region points for feature reconstruction. Feature reconstruction is implemented with solid modeling techniques (sweeping, skinning, covering etc.) and parameters are defined to provide editability of the features. We present examples and discuss extensions to our suggested framework.

CHAPTER 1. INTRODUCTION

1.1. Overview

Reverse engineering is a general concept that can refer to various fields, such as software management and product development [46]. In computer-aided design (CAD), reverse engineering aims to analyze a real object to determine its characteristics and mechanisms, with further focus on reconstruction and manufacturability. The data concerning the physical object can be obtained by various methods. A common method is using a 3D laser scanner or photogrammetry methods to obtain a point cloud corresponding to points on the surface of the scanned object. In the context of computer-aided design, reverse engineering is the process of obtaining a geometric CAD model from measurements acquired by scanning an existing physical model [89]. Reverse engineering is vital for various industries because the computer models acquired help improve the quality and efficiency of designs and also speed up the manufacturing and analysis process.

Reverse engineering techniques are traditionally used for the reengineering of machined parts. They are widely used in mechanical part engineering and manufacturing for re-engineering or replicating existing parts for which no CAD models exist. Also, there are cases where the original CAD model no longer corresponds to the physical part that was manufactured because of subsequent undocumented modifications made after the initial design stage. In objects of mechanical design, the geometric relationships of the component parts are well-defined and many characteristics of the object can be used in the reconstruction process. Properties of the object such as symmetries, parallelism and perpendicularity, can be used as attributes and constraints to aid the reconstruction process. Mechanical parts that are re-engineered must be accurate, robust and well-defined because they are usually combined and fit together with other components in larger objects.

Reverse engineering objects of freeform design is a more difficult task. This can be attributed to the fact that design intent and semantics cannot be strictly defined by geometric rules and constraints, as in mechanical part design. Freeform designs are often met in industrial design, such as automobile exterior parts design. Stylists and artists very often create physical models of their concepts by using clay, plaster or wood. These real-scale models are then used in re-engineering applications to create CAD models for manufacturing the objects on an industrial scale.

An interesting application of reverse engineering objects of freeform design is that of jewellery reconstruction. Jewellery design falls under the category of conceptual and decorative design. We can distinguish two categories of jewellery: free form jewellery and jewellery that conforms to certain patterns and constraints e.g. repeated patterns or specific gem cuts. Reverse engineering jewellery requires that the CAD models created are accurate and robust. These models should be parameterizable to support custom jewellery design. Furthermore, the user-designers should have the capability to modify the re-engineered CAD model according to their preferences, to create novel designs.

The aim of this work is to introduce a framework for reverse engineering objects of mechanical or freeform design to obtain fully editable feature-based 3D CAD models that can be reproduced or modified before production. More specifically, our approach aims to partition a 3D point cloud into components corresponding to the features, so as to create an editable and parameterized CAD model described by its various connected features. 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 the features of the model [41].

The framework of our approach is summarized in Figure 1.1. We begin from a 3D point cloud which we have preprocessed to remove duplicate points and an STL (stereolithography) file of the point cloud. An STL file is a file describing the model as a triangular mesh. This file format provides the vertices of each triangle of the mesh and its corresponding normal vector. We perform segmentation of the point cloud into subsets by detecting features using a point-wise characteristic called point concavity intensity and a local surface normal vector. We apply a region growing method based on variations of point concavity intensity and the surface normal to divide the point cloud into feature regions and regions corresponding to their boundaries. The boundary regions are approximated with cubic rational Bezier curves through a fast and efficient linear curve approximation method that we have developed. The resulting contours are used in combination with the feature regions to reconstruct each feature using solid modeling operations. The resulting features are then combined to construct the final 3D CAD model.

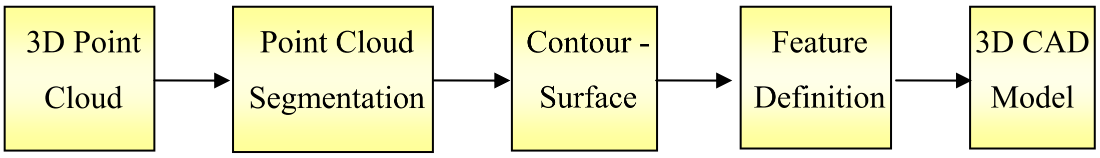


Figure 1.1 Our Feature-based Reverse Engineering framework

The feature-based reverse engineering approach presented in this thesis can be applied to point clouds of objects of mechanical or freeform design. This thesis makes the following technical contributions:

- Presents a segmentation method that partitions a point cloud into subsets that will be refined to correspond to features of the object being re-engineered,
- Introduces the concept of point concavity intensity, which is used for feature detection along with normal variance,
- Presents an efficient method for computing the concavity intensity of the point cloud.

- Introduces a fast linear curve approximation method for fitting cubic rational Bezier curves to 3D points corresponding to the borders of features
- Describes a methodology for reconstructing the features of the object into an editable 3D CAD model.
- Validates our approach using a real-world application (reconstruction of pierced jewellery).

This thesis consists of eight chapters. The rest of this chapter provides some preliminaries on scanning techniques and triangulation methods. In Chapter 2 an overview on reverse engineering approaches is offered. Work on reverse engineering is reviewed by focusing on the application scope and the type of CAD model representation used. In Chapter 3 we present our approach to detecting and extracting features from a 3D point cloud based on point concavity intensity and surface normal variation. A fast and efficient curve approximation method for fitting cubic rational Bezier curves to 3D points is presented in Chapter 4. Chapter 5 focuses on reconstructing the features from the feature regions using the contours constructed by the method provided in Chapter 4 and we discuss the editability of our features. In Chapter 6 we present examples of our framework whereas in Chapter 7 we examine extensions of our suggested framework and application to which our framework may be applied. Chapter 8 provides conclusions.

1.2. Preliminaries

Reverse engineering typically consists of four phases: data acquisition and preprocessing, data segmentation, surface fitting and CAD model construction. In the context of our work we focus on point cloud segmentation and CAD model creation. Therefore we will briefly present basic data acquisition techniques and triangulation methods used for the initial processing the scanned data.

1.2.1. Scanning techniques

The first step in reverse engineering is the acquisition of the data. [89] provides an analytical survey of data acquisition techniques. Data can be acquired from the surface of an object meant for re-engineering either indirectly, with non-contact methods, or directly, using contact methods. Non-contact methods use light, sound or magnetic fields to obtain surface data information. On the other hand, tactile (contact) methods use mechanical probes located on the end of a mechanical arm to actually touch the surface of the scanned object. Coordinate measuring machines (CMM) are the more popular tactile method because they can be programmed to follow specific scanning paths and the data they produce are very accurate and practically noise-free.

Optical methods are the most frequently used non-contact method. There are various approaches to using light sources for acquiring position data on an object's surface. A fast and common method is triangulation, where the locations and angles between light sources and photo sensitive devices are used to determine positions on the surface of the scanned object. A light source, i.e. laser, is projected onto the object's surface at a pre-specified angle and a photo sensitive device determines the angle of reflection off the surface. Triangulation is then used to determine the position data. Ranging methods are also frequently used, where the distance measurements are made by sensing time-of-flight of light beams.

These scanning techniques produce raw data. By raw data we mean an unstructured collection of geometric primitives such as a point cloud or a range image. The density of the data sets produced by the various methods depends on the sampling rate used to acquire information from the object's surface. Also, very often the point clouds obtained contain noisy data due to physical characteristics of the object, such as topology and texture, or limitations and regulations of the acquisition method used (i.e. accuracy limitations, calibration). However, processing methods have been suggested that can handle and overcome this problem.

1.2.2. Triangulation methods

The initial data representation of a scanned object is the 3D point cloud. From this form, models using other representation schemes can be constructed, depending on the application's aim and the intended use of the models. A standard data representation model that can be generated and useful for applications is the triangulated mesh. The triangulation of the point cloud provides basic connectivity and neighborhood information for each point of the raw data set and is usually used as an intermediate representation, before surface fitting and reconstruction is performed.

Many algorithms have been suggested for triangulating 3D point clouds [11]. [20] presents an incremental approach to building a triangulation by adding points based on specific criteria. [43] present an algorithm based on the idea of determining the zero set of an estimated sign distance function. Work such as [5] deal with the problem of point cloud triangulation from a computational geometry point of view, basing their approaches on variations of the Delaunay triangulation. In [45] a surface-based algorithm is suggested where for each point of the cloud, nearest neighbors and their orientations (in the estimated local tangent plane) are determined and this information is used in a growing process where a triangular mesh is constructed incrementally starting from a seed triangle by adding points based on the neighbor information.

1.2.3. Point cloud preprocessing

Point cloud preprocessing is often needed in the reverse engineering process to remove noise from the acquired data [89]. [37] perform filtering on the point cloud before using a curvature-based approach for data segmentation. Often smoothing techniques are applied to smooth out noise from the point cloud. Algorithms have also been developed for mesh smoothing and denoising [79]. These methods look to preserve the features of the object while removing noise from the mesh. A classical approach is to use some form of Laplacian smoothing [91].

Preprocessing is also used to detect symmetries that are helpful later on in the reconstruction phase. Such work is presented in [61]. The authors find approximate symmetries in a point cloud, since most of the time the data acquired by scanning contains errors that prevent the detection of exact symmetries.

CHAPTER 2. OVERVIEW – REVERSE ENGINEERING AND CAD

2.1. Introduction – Reverse Engineering and its applications

Reverse engineering is the process of obtaining a geometric CAD model from measurements obtained by scanning an existing physical model. The reverse engineering process consists, in general, of four main phases: data capture and preprocessing, segmentation, surface fitting, and the CAD model creation [89], [46],[14]. The first phase concerns the object data collected by some kind of scanning method and any type of preprocessing that may be carried out (e.g. noise removal). The segmentation and surface fitting stages refer to the decomposition of the original data set into smaller sets so as to fit surfaces to the points, thus leading to the last phase of the process which concerns the CAD model creation.

Reverse engineering methods are used in various applications. They are widely used in mechanical part engineering and manufacturing for re-engineering or replicating existing parts for which no CAD models exist. Also, there are cases where the original CAD model no longer corresponds to the physical part that was manufactured because of subsequent undocumented modifications made after the initial design stage. Works such as [84], [85] have concentrated on creating high accuracy models of manufactured mechanical parts. A characteristic of the objects re-engineered is that they are usually parts of larger objects and therefore have to fit and connect exactly with other parts, like pieces in a puzzle. For this reason, the models created through the reverse engineering process must be very accurate and well defined.

Reverse engineering is applied in industrial design, such as automobile exterior parts design. Stylists and artists very often create physical models of their concepts by using clay, plaster or wood. These real-scale models are then used to create CAD models for manufacturing the objects on an industrial scale. Also the CAD models provide the artists and stylists with the ability to re-evaluate their designs, especially when they can easily re-design or modify them as needed. Reverse engineering encourages conceptual design because the designer creates an initial prototype, scans it and manipulates it as desired. Reverse engineering is needed for aesthetic design because designing with CAD systems is quite challenging since many freeform surfaces are involved.

In addition, reverse engineering is used for the generation of custom fits to human surfaces and for mating parts. Examples of such applications are custom helmets, space suits and prosthetic parts. An example of such an application is described in [6], where generic models of mannequin torsos are fit to 3D point clouds of human torsos for garment modeling applications.

Reverse engineering techniques are often applied to medicine and animation. An example of a medical application is the creation of bone pieces to be used in orthopaedic surgery for the substitution of a shattered bone. The bone supplement is re-engineered to exactly fit with the neighboring bones of a specific patient. In animation, a prototype of a character is drafted, scanned, recreated from the 3D point cloud and finally used in an animated sequence.

Re-engineering objects of freeform design is essential for supporting custom design in a CAD model reconstruction system. It provides user-designers with the capability to modify re-engineered CAD models according to their preferences and to incorporate in novel designs. For instance, in the case of jewellery re-engineering, the user-designer might like to be able to modify the dimensions of a ring to produce one of larger size, or be able to choose certain parts of the object to use them to create other pieces of jewellery, e.g. a matching set of earrings. To this end, one needs to exploit the features of the original model and the relationships and constraints that hold among them.

2.2. CAD model representation schemes used in reverse engineering

Reverse engineering results in the creation of CAD models of physical objects. The more information a CAD model contains and provides the better, especially in cases of applications that demand accuracy and robustness. An appropriate CAD model should be able to capture design intent. A means to achieve this is by parameterizing dimensions and tolerances and defining constraints. Besides, a usable model need not necessarily result from exact interpolation of the 3D point cloud, but usually results from suitable approximations, since the point cloud most often contains noisy data.

Raw point data collected by 3D scanning techniques are usually rendered by building an interpolating robust polygonal mesh. This approach is accurate and fast but provides no means for large scale subsequent modifications. Only local interactive or non-interactive tools are provided that are usually targeted to correcting small imperfections and eliminating noise effects. CAD applications require robust and editable CAD models to support processes such as reproduction, design modification and redesign.

The type of representation scheme used for the model reconstruction in reverse engineering is heavily linked to the intended application. Some of the most often encountered CAD representation models are: point clouds, meshes, boundary representations (B-reps), constructive solid geometry (CSG) models, volume models and feature-based and constraint-based models. We will briefly refer to each model representation from a reverse engineering point of view.

A point cloud is a collection of 3D points that are acquired during the scanning phase of the reverse engineering process, and it is the initially obtained representation of the object. The point cloud describes the object without providing any information about the connectivity of the points, the geometry of the object or the design intent.

Therefore this type of representation model is inappropriate for applications where editability is required. Such representations are also usually costly in reference to system resources, especially in the case of large point clouds.

Meshes (e.g. polygons or triangles) lead to models that conform to the original physical object. However, meshes that perfectly interpolate the 3D point cloud via triangulation or similar methods do not capture design semantics, such as design intent, functionality and behaviour. Therefore, this type of model representation is not appropriate for objects with specific conceptual design and functionality, such as mechanical and industrial parts. Higher level polyhedral representations are not appropriate for describing complex and detailed objects. In this case, a large number of polygons is needed to sufficiently approximate the initial object and this is costly both time-wise and space-wise. Also, arbitrary shape manipulation is not efficient when the object is represented with polygonal meshes. The creation of accurate models is extremely difficult, especially when the object is small, complex and curvaceous, such as jewellery. This type of representation is suitable for rendering, not for interactive modifications and therefore very often other representation schemes are converted to polyhedral representation for this purpose. However there are reverse engineering approaches based on meshes. For instance in [88] the authors present an approach for manipulating triangular meshes to segment the mesh into feature regions and their connecting border sets. [11] presents an approach for using alpha shapes and simplified meshes for object reconstruction.

Boundary representation models (Brep) describe the edges and facets of the boundary of the object. It is the representation that naturally follows the point cloud phase during reverse engineering. Many algorithms have been developed for creating a boundary representation model from a 3D point cloud [10]. A Brep model may be constructed using NURBS (Non-Uniform Rational B-splines) or other surface patches. This type of representation is useful in applications where freeform surfaces are part of the repertoire of primitives [94]. Brep is useful for representing any type of object, such as mechanical parts and objects of aesthetic design. Surfaces can be described using appropriate parametric representations. However, surface patches and plain Brep models do not capture design aspects of the object that refer to functionality and part relationships. Therefore, the information provided through this type of model is limited and does not provide tools for modifying parts of the model that affect the whole design (usually editing of local features is feasible by interactively placing control points). However, Brep models are fundamental representations which can be used in combination with other elements (e.g. features, constraints) to achieve more flexible and useful models. For instance [53] creates a surface representation from a point cloud. The surfaces are grouped into feature objects and then the model is modified based on constraints derived from regularities that exist in the Brep model, to achieve a more ideal representation of the original object.

Constructive Solid Geometry (CSG) models are created by performing Boolean operations on solid primitives e.g. spheres, cones, cylinders and cubes. From this definition we can perceive that CSG models can only represent objects that can be created from solid primitives, therefore this occludes the representation of free-form surfaces and objects. In general, the CSG model representation can be used in mechanical part engineering and manufacturing applications or in other applications where the design history of the objects can be recorded as a sequence of Boolean operations on geometric primitives. More abstract designs cannot be represented by this type of model. Converting CSG models to render-able ones is extremely difficult and therefore CSG is commonly used in conjunction to Brep. In this case a Brep model is always maintained and every modification is transformed to an incremental Brep editing operation.

Another scheme that can be used in reverse engineering is the voxel-based approach. The object being reverse engineered does not necessarily pass through the 3D point cloud phase, but can be initially represented directly as a volume model made up of voxels. These models can be created using i.e. haptic shape modeling techniques such as in [97]. This approach to reverse engineering is useful for avoiding the 3D point cloud data phase, which is cumbersome because of the need to remove noise, and for creating robust models. However, it is an approach feasible only in the case where we are allowed to come into contact with the object being scanned and our probe is appropriate for the level of complexity of the object. Voxel-based representations are useful in representing solid objects, because they provide information about the volume properties of the object. However, much like CSG models, they are not appropriate for object models that are to be modified or manufactured because they are not “flexible”. Editing specific parts of the model cannot be carried out unless the model is transformed into a surface model first. Also, relationships between parts of the object are not defined, nor is it clear how each part is connected to others. In our context, voxel-based models are useful if they are combined with feature and constraint properties. The voxel-based representation can also be used as an intermediate representation during surface reconstruction such as in [92].

A model representation that is growing more and more popular is the feature-based and constraint-based model. The model is described by its features and the relationships between them [71]. Constraints are applied to the features to create more accurate and robust models but also for beautification. This type of model representation is known to be appropriate for manufacturing of mechanical parts, where there are well defined relationships among the different elements of the model. Also, feature-based models are well suited to industrial design and manufacturing since the model can be easily modified. This is due to the knowledge provided by the model concerning tolerances, constraints, relationships and connectivity among the features. Therefore, feature-based methods are often characterized as knowledge-based [31]. Their main objective is to exploit any knowledge and information that is connected to design intent, functionality and construction process of the object being re-engineered. We will examine these types of models more thoroughly in the following section.

An object can also be represented by its skeleton. By skeleton we mean the closure of all points that have more than one closest point on the shape boundary (for example the medial axis transform). This representation provides the topology and shapes that exist in the object and also reflects the symmetries of an object. Depending on the type of application the skeleton is used for, it may be a 2D or 3D representation. For instance, in 3D the medial axis transform produces a medial surface. The exact computation of the 3D skeleton is a computationally intensive problem that returns a skeleton as complex as the object itself. Therefore we usually seek for an approximation. A skeleton representation scheme is used in various CAD applications for object recognition and retrieval [22], animation [13] and other solid modeling operations([72, 78]). It is widely used in feature-based modeling, where it can be employed to describe the shape of features, in feature detection and extraction applications and shape deformation, for instance refer to [58] and [99].

2.3. Using high-level model representation schemes for reverse engineering

Applications that call for an acceptable aesthetic result as well as the capturing of the design intent, i.e. industrial and automobile design, make the use of constraints mandatory for a satisfactory aesthetic result. These types of applications are well met using higher- level representation schemes (i.e. constraint-based and feature based models) that incorporate knowledge and constraints into the model to make it more flexible and editable. Also complex applications, such as rapid prototyping, model redesign and reproduction call for editable 3D CAD models that are smooth, robust, and accurate. A characteristic of the objects re-engineered is that they are usually parts of larger objects and therefore have to fit and connect exactly with other parts, like pieces in a puzzle.

For editable models that can be used in redesign and custom design, feature-based approaches in combination with constraints is preferred so that local and global modifications can be made on the model. Also if the model is represented as a union of features that are defined by parameters then these feature objects can be used in other models. A simple representation of an object using surface patches makes it very difficult to perform global or even local modifications on the object without compromising the robustness of the model. Higher level representations such as feature-based models allow modifications to be propagated throughout the model without affecting its robustness and accuracy. We will briefly review the projects that have focused on using this type of model representation for reverse engineering.

Research such as [84, 85] have concentrated on creating high accuracy models of manufactured mechanical parts. The REFAB project [26, 85] uses a feature-based and constraint-based method to reverse engineer mechanical parts. REFAB is a human interactive system where the 3D point cloud is presented to the user, and the user selects from a list a feature that exists in the cloud, specifies with the mouse the approximate location of the feature in the point cloud, and the system then fits the specified feature to the actual point cloud data using a least square means method iteratively. The authors give emphasis on the fitting of pockets, where the user draws a profile of the pocket on the point cloud and the system then fits the profile to the data and the profile is then extruded to create the pocket. This feature-fitting process is made more accurate by using constraints that are detected by the system, verified by the user and then exploited to achieve a better fitting of the features according to the data. The system supports constraints such as parallelism, concentricity, perpendicularity and symmetry. The constraints defined and used in REFAB seek to reduce the degrees of freedom associated with the object as much as possible, so as to achieve high precision models in less time.

A feature-based reverse engineering method was also used by Au et. al [6] for reverse engineering a mannequin for garment design. Generic models of mannequin torsos are fit to 3D pint clouds of human torsos for garment modeling applications. The basic concept in this method is to create a generic mannequin model of a human torso, which is appropriately aligned with the 3D point cloud of the desired human torso model, and the generic model is fit to the point cloud by matching up characteristic points of the models e.g. peaks. This method creates parameterized models by exploiting the features of the object and by using them to constrain the fitting process. It is an automated approach to reverse engineering human torsos that creates parameterized models with good accuracy.

Work such as [52, 53] concentrate on how constraints can be detected and efficiently applied in the reverse engineering procedure to create more accurate and aesthetic models. The authors analyze the type of symmetries and shape regularities that can be observed and detected in a brep model and how they can be grouped into constraints that can be applied on the model. [9] focus on constrained fitting and how constraint systems can be more efficiently solved and applied to achieve a better model.

[88] presents a reverse engineering framework where a mesh is segmented based on techniques derived from morse theory. The mesh created from the point cloud is divided into separator sets which are combined with feature skeleton to detect primary regions of the object which are finally fitted with surface patches.

In a nutshell, research on extracting and exploiting features from point clouds of objects of both mechanical and freeform design for the purposes of redesign, reproduction and custom design is still ongoing and a topic of current interest. In this context, we present an approach that detects and defines features and combines them with solid modeling techniques to achieve robustness and editability.

CHAPTER 3. FEATURE DETECTION AND EXTRACTION

3.1. Introduction

In this chapter we present a method for detecting and extracting features from point clouds for the purpose of re-engineering. The objects that can be re-engineered by the proposed approach can be either of mechanical or freeform design. Re-engineering objects of freeform design is relatively more difficult and complex than reconstructing mechanical parts. Mechanical parts usually have specific geometric characteristics, such as symmetries and swept profiles that are fairly easy to detect and parameterize. On the other hand, in the case of freeform objects, there are often features that are difficult to extract. In the case of restricted mechanical parts, features libraries can be defined and used for detecting features in the point cloud, whereas for freeform objects this is not feasible.

3.2. Related Work

The first issue we face in creating robust feature-based CAD models from 3D point clouds is that of segmenting the point cloud into individual subsets that correspond to features. Much work has been done on segmenting a point cloud into feature-like subsets for surface reconstruction. The usual approach is to exploit the changes in the surface curvature and to follow a region growing approach. In [88] a method for detecting and extracting feature regions in a point cloud is presented based on hierarchical morse complex segmentation and feature skeleton. Primary regions and separator sets (blending regions between primary regions) are detected and surfaces are fit to these regions. Another approach to segmenting a point cloud is presented in [27] where the authors detect closed sharp feature lines as borders of surfaces patches by applying a region growing method with normal estimation to produce clusters of points that, when represented as a graph, can reduce the problem size by representing the point cloud. However this approach is limited to objects with sharp features, and therefore it is not applicable to objects of freeform design. [45] focuses on a curvature based approach to feature line extraction from meshes of point clouds of mechanical parts.

On a different level, in the REFAB project [26], a user-computer interaction approach is followed for data segmentation. Specifically, the end-user specifies modeling characteristics in the point cloud, such as profile curve and points for pocket construction, and the system fits the feature into the point cloud.

We propose a method that achieves a feature-based segmentation of the point cloud by using the concavity intensity of the point cloud to decompose it into subsets (components) that correspond to features of the physical object. With the term “concavity intensity” we refer to the smallest distance of a point of the cloud from the convex hull of the cloud. The sets of points are then fitted with curves and surfaces, and feature attributes and constraints for each component are derived. The CAD model is then built on these components by defining and applying constraints concerning the features and their connectivity.

In the following we will present our point cloud decomposition method, which combines point cloud concavity intensities with region growing.

3.3. Deriving Point Concavity Intensity

Throughout the following we assume that our point cloud has been pre-processed to remove duplicate points (Figure 3.1). The first step in performing feature-based reverse engineering is to divide the point cloud into individual components, such that each component corresponds to one or more features.

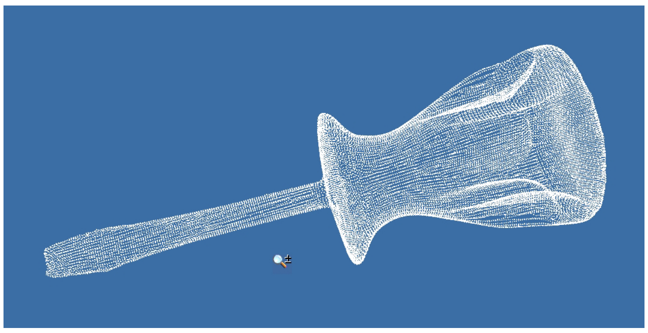


Figure 3.1: Point cloud of a screwdriver with 27k points [23]

The work presented in this section employs a characteristic introduced in [57, 58]. The authors present a shape decomposition and skeletonization method for polyhedrons that is based on approximate convex decomposition. The convex hull of a polyhedron is computed and the concavity of the vertices of the polyhedron is calculated and used as a criterion for the decomposition of the object [56]. The concavity of a vertex is defined as the distance from the vertex to the convex hull surface of the component/polyhedron. The polyhedron is split into components at locations where the concavity of the vertices is high. An iterative method is then used to simultaneously find the most efficient shape decomposition and the skeleton of each decomposed component. We apply the principle of vertex concavity in conjunction with normal variation to decompose point clouds into components that represent features.

We compute the 3D convex hull of the point cloud using the Quickhull algorithm [7] (Figure 3.3). We then calculate the concavity intensity of each point of the point cloud.

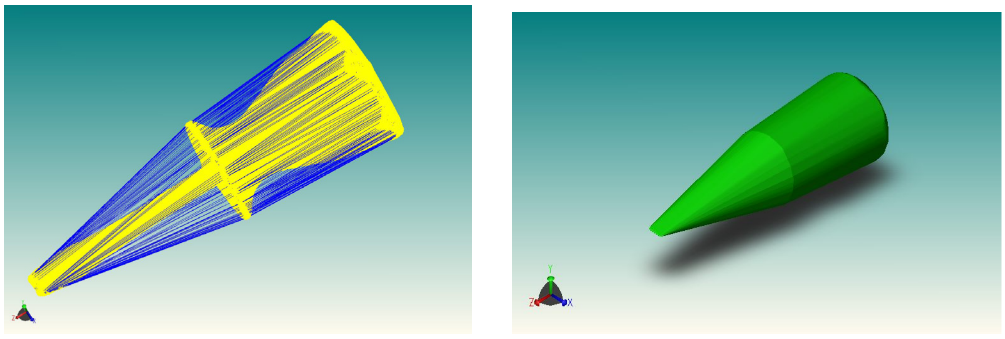


Figure 3.2: A point cloud of a screwdriver point cloud and its convex hull

We define the *concavity intensity I(p)* of a point p as follows. Let p_t be the track of the point on the convex hull on some face *f* of the convex hull. Then *I(p)* is the smallest distance $||pp_t||$ such that the line segment pp_t does not cross the cloud point.

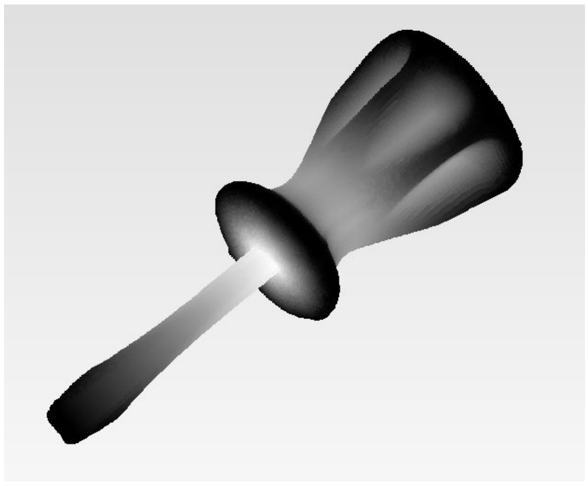


Figure 3.3: The screwdriver point cloud concavity intensity map [23]

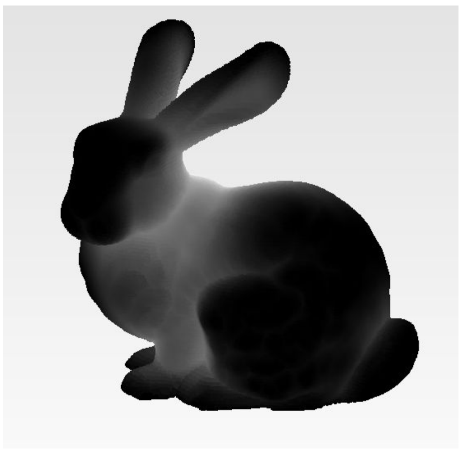


Figure 3.4: The Stanford bunny concavity intensity map [77]

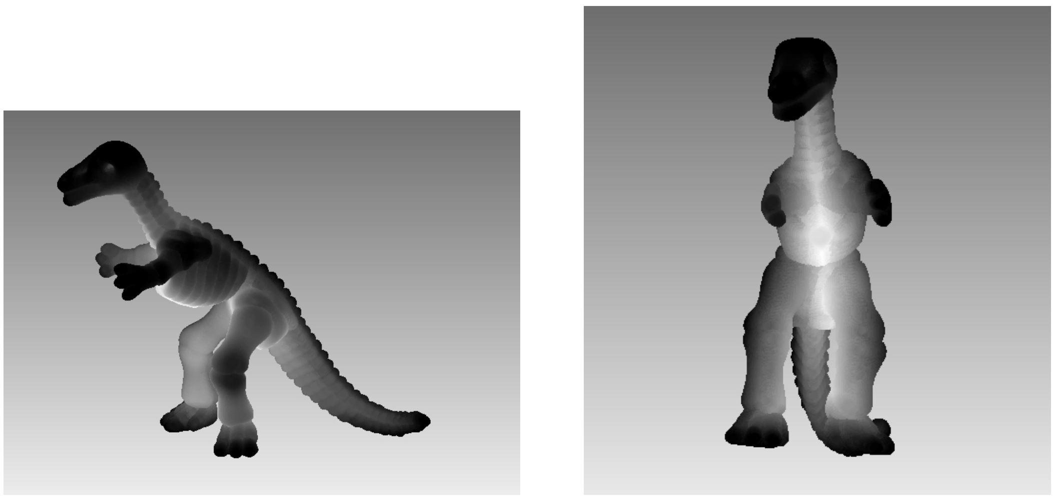


Figure 3.5: Concavity intensitiy map of points representing a dinosaur [23]

Figure 3.3, Figure 3.4 and Figure 3.5 display the concavity intensities of points belonging to different point clouds, a screwdriver, a dinosaur [23] and the Stanford bunny [77]. In the concavity intensity map the values are rendered using greyscale, where black corresponds to points belonging to the convex hull, whereas white corresponds to points that are farthest away from the convex hull. We can observe that edges (rapid variations in concavity intensity), saddle points and extrema can be used to partition the point cloud into components that can later be refined as the features of the object.

A brute force approach to computing concavity intensity of each point of the point cloud is to calculate the distance of each point from each facet belonging to the convex hull of the point cloud. Then we check for intersection with the triangulated point cloud and select the smallest distance derived by a projection that does not intersect the cloud point. This however results in general in a time complexity of O(n^2), where n is the number of points in the point cloud. This is prohibitive for large complex point clouds with convex hulls that consist of many facets. Therefore, we have developed an algorithm for computing the intensity of each point without having to examine all the polyhedra of the convex hull. More specifically, it is evident there are convex hull facets for which it is meaningless to examine their distance from the point cloud point, simply because topologically, in reference to the point, they are located much farther away than other facets. For instance, suppose we would like to measure the distance of a point close to the tip of the screwdriver from the convex hull. Thus, the facets belonging to the convex hull that are located far away from the point under consideration cannot contribute to the smallest distance from the convex hull. The target facet will be located somehow close to the point.

The search algorithm starts by calculating distances from facets containing a specific vertex v belonging to the convex hull. The vertex is chosen based on its distance from the point p in one of the three coordinate directions x, y and z. Let $S(p, r)$ be a virtual sphere with radius r and center p and apply the search algorithm for each convex hull vertex that is located inside this sphere (Figure 3.6). The radius r is calculated by determining the smallest difference (p_x-$v_x \neq 0$) between coordinates along one of the direction axes, e.g. along the x axis, and multiplying it by constant accuracy parameter k, which defines the final size of the virtual sphere. A larger sized sphere corresponds to more convex hull vertices for which the algorithm will be applied.

$$r = k(p_x - v_x) \qquad \text{(Eq. 3.1)}$$

The radius *r* is estimated after a short pre-processing step applied to the convex hull vertices. We sort the vertices individually for each coordinate direction x, y and z (O(nlogn) pre-processing time) and locate the closest axis-wise neighbours in logarithmic time (O(logn) using binary search).

To enforce the non-crossing requirement we apply an additional constraint for choosing vertices of the convex hull. Specifically, for each of the points located inside the virtual sphere with radius *r*, we only apply the algorithm for vertices belonging to facets that are located on the correct "side" of the convex hull, in reference to point *p*. The "correct side" of the convex hull is determined with the help of the normal vector of the local surface at *p*, which is calculated from the adjacent facets to p (note that we have performed a triangulation of the point cloud using Tight Cocone in O(nlogn) expected time [28]. From the triangulated model *T* of point cloud we determine the neighbouring points of each point *p* and the normal vector n_i of each triangle adjacent to *p*. By computing the vector sum of all the normal vectors of the neighbouring triangles, we obtain an estimation of normal vector of the local surface at *p*. This is then used to decide which points of the convex hull are on the appropriate side of the surface.

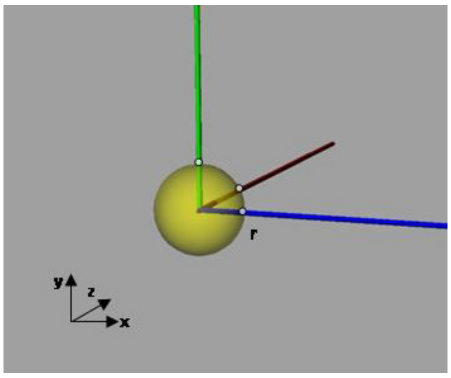


Figure 3.6: The virtual sphere S(p,r)

The search algorithm is a recursive procedure that performs a local search around the vertex *v* to locate the convex hull facet for which the distance from *p* is minimized. For this facet the algorithm is repeated until no facets with smaller distances are detected.

The algorithm takes as input the point cloud *S*, a triangulation *T* of the point cloud and the convex hull *H*. For each point *p* of the point cloud, we first determine if it belongs to the convex hull. In this case, the concavity intensity of the point is zero (*I(p)*=0). If the point does not belong to the convex hull then we must find the facet *f* of the convex hull for which the distance between point *p* and the track *p'* of *p* on the plane *f* is minimum ($d_{min}(p,p')$). This minimum distance is determined by applying a recursive local distance search which begins at a convex hull vertex *v* and, after determining the distances of point *p* from each convex hull facet *f* that has *v* as a vertex, searches the remaining vertices of the facet with the smallest distance. This process is repeated every time a facet is found for which the minimum distance(s) is(are) smaller than the previous detected.

Figure 3.8 displays an example of the local search algorithm. By applying the starting point selection criteria, we determine that a possible starting point of the convex hull for the algorithm is point #2. The minimum detection process begins by calculating the distance of point *p* from each adjacent facet to point #2. From the calculated distances we conclude that the smallest distance is located for facet (2, 4, 15). The algorithm then continues by repeating the above process for vertices #4 and #15, for which the smallest distance is returned for facet (4, 15, 16). The next repetition of the algorithm returns that none of the remaining facets formed by #16 correspond to distances smaller than that returned by facet (4, 5, 16). Therefore the algorithm terminates and returns the intensity of point p which is *I(p)*=0.000014.

This approach to computing the concavity intensity of a point is guaranteed to compute the correct distance since it is performed on the convex hull. The shortest distance cannot correspond to a projected point outside the convex hull; the shortest distance always corresponds to a projection that is located inside a polygonal facet of the convex hull. This is demonstrated using an example in 2D space. Suppose a point *p* of the point cloud is projected on edge *f* and let *p'* be the closest point on the line defined by *f*. Then, ||*pp'*|| is the corresponding distance (Figure 3.7). If *p'* is outside the line segment *f* then *pp'* should intersect a neighbour edge f' at point q. But then ||*pp*'|| is larger then ||*pq*|| which contradicts our initial assumption. Thus *f* is not the closest edge to *p* and we continue by taking the projection of *p* on *f'* finding *k* which in this case is the closest point of the polygon to point *p*.

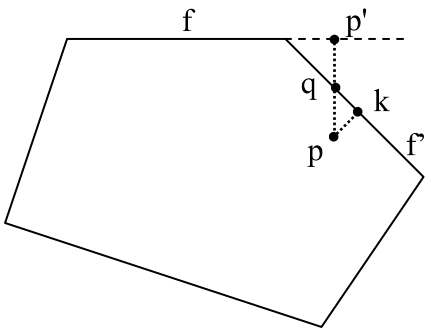


Figure 3.7 Finding the closest edge to point *p*

The point concavity intensity calculation is summarized in Algorithm 3.1 and Algorithm 3.2.

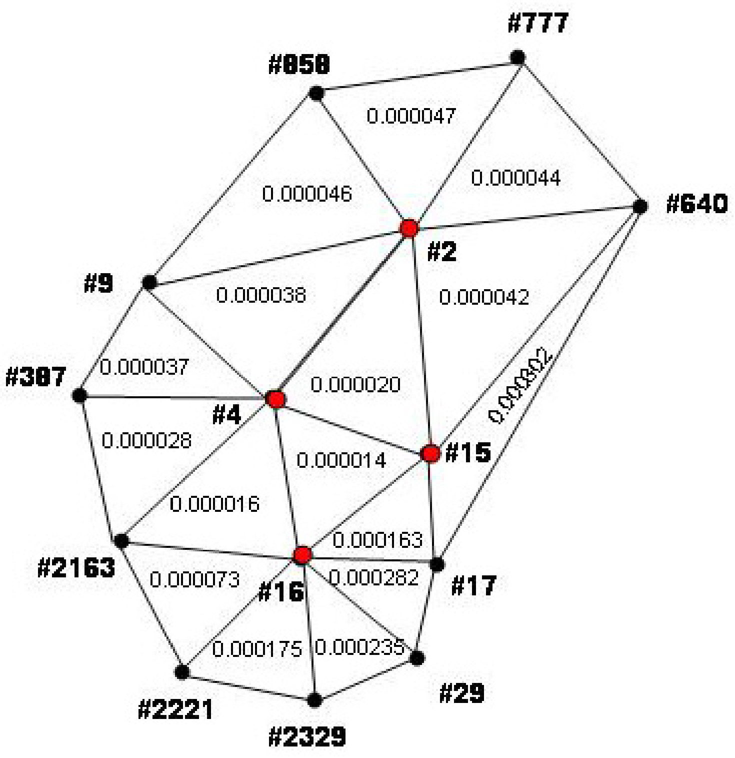


Figure 3.8: An example of the local distance search algorithm

Algorithm 3.1: Local Distance Search Algorithm

```
Local_distance_search ($s$, $p$, $d_{pre}$)
Input: Starting point $s$, cloud point $p$, previous smallest distance $d_{pre}$
Output: Minimum Distance $d_{min}$
1.  For every facet $f_i$ $(s,s_1,s_2)$ of $s$
2.      find projection point $p'$ of $p$ on the plane of facet $f_i$
3.          if $d(p, p') < d_{pre}$
4.              $d_{pre}= d(p, p'), v_1=s_1, v_2=s_2$
5.    end
6.    if $d_{pre} < d_{min}$   then   $d_{min}=d_{pre}$
7.    $d_1$=local distance search($v_1$, $p$, $d_{pre}$)
8.    $d_2$=local distance search($v_2$, $p$, $d_{pre}$)
9.    if $d_1 < d_{min}$   $d_{min}=d_1$
10.   if $d_2 < d_{min}$   $d_{min}=d_2$
11. return $d_{min}$
```

Algorithm 3.2 Point Concavity Intensity Computation

```
Point_concavity intensity ( )
        Input: Point cloud S, triangulated mesh T of S
        Output: Concavity intensities $I_i$ for every point $p_i \in$ S
1.   For every point $p_i \in$ S
2.       if $p_i \in$ convex hull
3.       return $I(p_i)$=0
4.       else
5.           find the starting vertices $v$ of the convex hull that are located inside a
             virtual sphere with radius $r$
6.       for every vertex $v_j$
7.           initialize $d_{min}$ and $d_{pre}$
8.           $d_j$=local_distance_search($v_j,p,d_{pre}$)
9.       end
10.      return $I(p_i)$=min($d_j$)
11.      end
12.  end
```

Algorithm 3.1 has been experimentally observed to run in constant time (worst case being O(n)) whereas Algorithm 3.2 runs in O($n$log$n$) time, where n is the cardinality of the point cloud.. The triangulation of the point cloud is obtained in O($n$log$n$) time, using TightCocone, the sorting pre-processing takes time O($n$log$n$) and searching for closest vertices of the convex hull is performed using binary search in O(log$n$) time. Thus the overall expected time complexity is O($n$log$n$).

3.4. Point cloud segmentation through region growing

The point concavity intensity values calculated in the previous section are used to segment the point cloud into feature components. A feature component is bounded by areas where abrupt changes in the direction of the normal and/or rapid concavity intensity variations are observed. After calculating the concavity intensities of all the points that form the point cloud, we apply a region growing method to divide the point cloud into its components.

Our region growing method is based on two criteria:
i) the normal vectors of neighbouring points belonging to the same region should form an angle smaller than a threshold t and
ii) the approximate gradients of the concavity intensity function in directions x, y and z for neighboring points of the same region should maintain the same sign value, meaning that there are no zero crossings observed between them.

From the triangulation of the point cloud we can derive a normal vector for every triangle belonging to the mesh. For every point of the point cloud we compute a mean normal vector by taking into account the normal vectors of all the triangles that have the point as a vertex. This mean normal vector represents in general the direction of the normal of the local surface area created by these facets at the specific point location. By computing the angle between the normals of two points we can obtain information about the object's shape at that location. For instance, if the normal vectors of two neighboring points form a sharp angle, then it is most probable that a sharp feature is located at that position in the original object. If the angle formed by the vectors is small, then the surface area is smooth, implying that the points belong to the same shape feature. Therefore, the first region growing criteria can be summarized in the following equation:

$$\alpha(norm_i, norm_p) \leq t \qquad \text{(Eq. 3.2)}$$

where α is the angle formed by the normal vectors belonging to points i and p, and t is a threshold value used to determine if the points belong to the same region or not. The threshold t is an adaptive factor that can be used to define the number of regions that the method will detect. Specifically, if the threshold is small, then the criteria takes on a more strict nature that leads to the detection of more regions, than if a larger threshold is used. A small threshold detects any anomalies and shape changes that may exist in the object, whereas a large threshold detects more intense changes in the shape of the object, thus leading to fewer regions.

Before applying the region growing method to our point cloud we perform a pre-processing step to calculate the variations in concavity intensity values (approximate gradients) for all the points in the point cloud for coordinate directions x, y and z.

The variations in concavity intensity values are determined by examining the concavity intensities of neighboring points. We will refer to neighbors directly connected to a point *p* of the point cloud as its first-level neighbors, whereas with the term second-level we will refer to the neighbors of *p*'s neighbors (points r_i - Figure 3.9). We calculate an approximate gradient of the intensity for each point *p* in coordinate directions x, y and z using the following equations:

$$dI_x = \frac{\sum_{i=0}^{n} \frac{I_p - I_i}{x_p - x_i}}{n} \quad \text{(Eq. 3.3)} \qquad dI_y = \frac{\sum_{i=0}^{n} \frac{I_p - I_i}{y_p - y_i}}{n} \quad \text{(Eq. 3.4)}$$

$$dI_z = \frac{\sum_{i=0}^{n} \frac{I_p - I_i}{z_p - z_i}}{n} \quad \text{(Eq. 3.5)}$$

where I_k is the concavity intensity of point $k(x_k,y_k,z_k)$ and n is the number of *k*'s neighbors. However, since the neighbors of a point may be so close that the coordinate difference in a direction is practically zero, i.e. x_p-$x_i \to 0$, we calculate the approximate gradients in respect to the second level neighbors of point *p*. Thus the above equations are applied for the second level neighbors, meaning that I_k is the concavity intensity of a second level neighbor k of *p* and *n* is the number of second level neighbors of *p*. The approximate gradients reveal how the intensity function of a point changes in every direction in reference to its neighboring points.

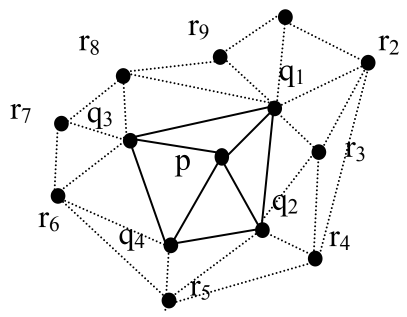


Figure 3.9: Point p and its first level neighbors q_1,q_2,q_3,q_4. The remaining nodes are second levels neighbors of p

After computing the gradients we apply our region growing method. As seed points of the method we choose points where the concavity intensity values are constant or almost constant. Region growing is carried out by adding points to a region if the two criteria are met. If a point does not satisfy the criteria, then it is most likely that it belongs to a border area around the growing region.

More specifically, the region growing method begins by examining every first-level neighbor of the seed. For each such neighbor *q*, we calculate the angle formed by the normal vector of the seed and the normal vector of the neighbor *q*, using the dot product. If the angle is smaller than a threshold t then we assume that the direction of the normal in the local area is maintained, thus the neighbor is added to the region. If the angle is larger than the threshold, we must take into account the concavity intensity gradients of the seed and the neighbor to determine if the point belongs to the same region, or if it belongs to a saddlepoint or extrema.

To ascertain the behaviour of the concavity intensity function in the area, we multiply the gradients of the concavity intensity function of the seed and its neighboring point in each coordinate direction. If $dIs_x \cdot dIq_x > 0$, $dIs_y \cdot dIq_y > 0$, $dIs_z \cdot dIq_z > 0$ this means that there is no change in the direction of the intensity function in these coordinate directions. Thus the neighbor belongs to the region and therefore it is added. If $dIs_x \cdot dIq_x < 0$, $dIs_y \cdot dIq_y < 0$, and $dIs_z \cdot dIq_z < 0$ then the sign of the gradient of the seed is different than the sign of the neighbor's gradient in the specific coordinate direction, implying that there is a zero crossing between the two. A zero crossing reveals the existence of a saddlepoint or an extremum.

If neither of the region growing criteria is met for point q, then we take into consideration the second level neighbors of q to ensure that q is a saddlepoint or extrema. More specifically, the first level neighbors of the neighbor q are examined in reference to the seed. The mean normal vector of the first level neighbors of q is computed and the angle between this normal and the seed normal is calculated. If the two vectors form an angle smaller than a more relaxed threshold t' and the approximate gradients of the first level neighbors of q is consistent with the behaviour of the region, then the neighbor is added to the region. In the opposite case, the point most possibly is a saddlepoint or extremum.

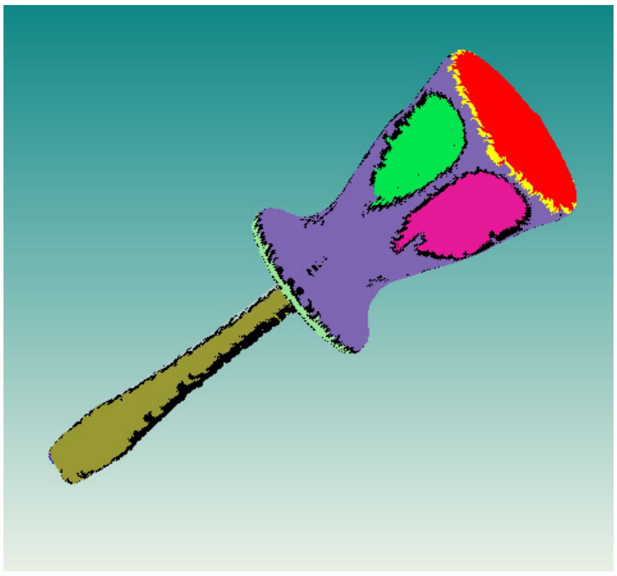


Figure 3.10 Feature regions detected by region growing

In Figure 3.10 the result of the region growing algorithm when applied to the screwdriver point cloud is shown. The region growing method has been implemented under the Microsoft Visual C++ programming environment using ACIS R18 solid modeling libraries by Spatial and HOOPS 16.20 for the GUI .

A short post-processing step on the feature regions may be performed manually after region growing. Specifically, we can merge two regions together to form a single region or we can split a region into two smaller regions. This is done manually by specifying interactively points which can form a border between the two new regions. Also we can create user-defined hardwired point-wise boundary contours to limit the behaviour of the region growing method, when recalculating the regions on a portion of the point cloud. Also border correction can be performed manually, in cases where a feature's border is not continuous, by defining connecting points that belong to the border.

The criteria used by the region growing method to determine if a point belongs to a region or not can be synopsized as follows:

A point p_i belongs to a region *r,* whose seed is point p, if:

1) the angle formed by the normal vectors of neighboring points p and p_i is smaller than a threshold t

$$\alpha(norm_{p_i}, norm_p) \leq t$$

2) the gradients of the concavity intensity function in directions x, y and z for p and p_i maintain the same sign value, meaning that there is no zero crossings observed between the two

$$dI_{p_i x} \cdot dI_{px} > 0,\ dI_{p_i y} \cdot dI_{py} > 0,\ dI_{p_i z} \cdot dI_{pz} > 0,$$

The region growing method is summarized in Algorithm 3.3.

Algorithm 3.3 Region Growing Method

```
Region_growing (s, r)
      Input: Seed s
      Output: Region r
1.    For each neighbor $i$ of $s$
2.      If angle $a(\text{norm}_i,\text{norm}_s) < t$ threshold then
3.        add $i$ to region $r$
4.      else
5.          if $dI_{sx}\cdot dI_{ix}>0$ and $dI_{sy}\cdot dI_{iy}>0$ and $dI_{sz}\cdot dI_{iz}>0$ then
6.            add $i$ to region $r$
7.          else
8.            for every neighbor $i'$ of $i$
9.              if angle $b(\text{norm}_{i'},\text{norm}_s) < t'$ and $dI_{sx}\cdot dI_{i'x}>0$, $dI_{sy}\cdot dI_{i'y}>0$, and $dI_{sz}\cdot dI_{i'z}$
                >0 then
10.               add $i$ to region $r$
11.             else $i$ belongs to a border region/saddlepoint
12.             end
13.           end
14.         end
15.     end
16.   end
```

CHAPTER 4. CONTOUR RECONSTRUCTION

4.1. Introduction

In the previous chapter we presented an approach for discovering features on a point cloud by detecting local variations in the morphology of the point cloud. This approach results in a number of regions that represent object features and regions representing the boundaries of the features. The boundaries of the features are approximated by a collection of piecewise cubic rational Bezier curves that best fit the detected border point cloud and are G^1 continuous. In general, we present a fast curve approximation method that approximates raw data with cubic rational Bezier curves. Our approach combines linear least squares approximation with continuity constraints to ensure G^1 continuity between neighboring curves. We use the weights of the curve to adjust its shape and parametric structure so as to construct curves that pass as closely as possible between the data sets and join smoothly.

4.2. Related Work

The problem of constructing curves that approximate point cloud data has been approached using different type and degrees of curves depending on the nature of the application. The most straightforward approach to curve fitting is to fix some curve parameters such as knots and weights (for Bsplines), and then use a least squares fitting approach to compute the control points of the curve [65]. [93] provides a thorough survey on curve fitting.

[98] presents a method for fitting rational Bspline curves to point data for reverse engineering applications. The authors suggest a linear least squares optimization process where the control points and the weights of a rational Bspline curve of degree *n* are iteratively refined until convergence is achieved. In [83] the authors focus on constructing curves and surfaces from point clouds obtained by 3D scanning techniques. The initial point cloud is triangulated and a method is introduced for selecting appropriate points from the data set based on the triangulation. Then a curve refinement process is performed which fits the Bspline curve to the points under linear constraints related to the endpoints and tangent vectors. [66] proposes a method based on squared distance minimization, which starts with an initial Bspline curve that is iteratively fitted to the target curve. In [93] the authors suggest a method for computing a planar Bspline curve to fit an unorganized, possibly noisy, point cloud using an iterative squared distance minimization process based on [66] which converges from an initial curve to the desired target shape using a squared distance error objective quantity. [54] suggests a modified moving least squares method for thinning out a point cloud and approximating it with a smooth curve. In [30] the authors present an approximation method which constructs smooth curves from noisy unordered data.

[87] discusses the pros and cons of constrained and unconstrained fitting in reverse engineering applications. As noted by the authors, unconstrained fitting results in root-mean-square distance minimization of the points, however fairness of the curve is not taken into consideration, which is important in re-engineering applications. Also, with noisy point cloud data, unconstrained curve fitting may lead to unwanted results. The authors provide a general curve fitting approach which, depending on what type of constraints are integrated into the process, achieves different fitting results. Constraints to trim the search space of the curve fitting process are reported in [33], where the authors use constraints to ensure that the curves lie on smooth manifolds.

While the method proposed in [98] produces curves for the purpose of reconstructing objects of freeform design, its drawback is that convergence is somewhat slow, making it costly to use for re-engineering objects consisting of many complex features. The curve fitting methods proposed in [66] and [93] are very accurate and stable, however their performance depends on the initial curve specified interactively by the user. [54] constructs a bspline curve to approximate a thinned out point cloud, without providing an upper bound for the approximation error. In [83], even though constraints are imposed on the curve fitting process, the curves obtained are not sufficient for representing complex shapes often encountered in freeform objects. The work in [30] is based on the assumption that no connectivity information is available, however, in our case, the point cloud has been pre-processed and therefore we have acquired topological information regarding the point cloud.

Software tools have been developed that can be used for curve reconstruction in reverse engineering applications. An example of such software is SISL [73], a NURBS software library that provides functionality for building applications handling freeform geometry. SISL uses a global optimization approach to curve fitting that returns very good results. We have implemented a global optimization for our application problem and observed that is usually converging quite slowly, and depending on the initial condition we may encounter accuracy issues and convergence to local minima. Our application problem deals with a multitude of features resulting from processing the 3D point cloud. The boundary regions of the features should be approximated by piecewise curves for purposes of reverse engineering and therefore require a effective and efficient method.

The work in this chapter focuses on re-engineering point clouds to construct feature-based CAD models that can be used mainly in redesigning and reproduction. We look to reconstruct not only mechanical parts but also objects of freeform design. For CAD data representing objects of freeform design an approach that yields reasonable results is the use of piecewise rational curves of low degree, because the weight factors allow the shape of the curve to be better adjusted to the point data by determining the effect the corresponding control point has on the curve.

4.3. Curve approximation using cubic rational Bezier curves

The region growing method mentioned in the previous chapter provides sets of points corresponding to borders of feature regions. Suppose we have a region border consisting of n 3D points. We would like to find the curve that best approximates this data set to represent the general morphology of the border. We require that the method used for fitting is of low computational complexity and of low degree. Approaches in the literature did not meet these requirements either because of slow convergence, or usage of curves of higher degree or no consideration of the fairness of the curve.

Our approach is similar to the work in [98]. In this work the authors present an optimization process through which rational b-spline curves are fit to point data. We adopt this approach to fit cubic rational Bezier curves to sets of points corresponding to feature boundaries and extend it to ensure that the curves created conform to the conditions required for G^1 continuity.

We use an equivalent instance of the general NURB, namely piecewise rational cubic Bezier curves because the constraints we apply 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 we basically follow an optimization approach which can inherently rule out noisy data without affecting the shape of the boundary as a whole. Rational curves are flexible curves that can approximate complex geometry more accurately than pure polynomials. In general they are not preferred for reverse engineering applications in the sense that their nonlinear multivariate format is not computationally practical. However they are not as expensive and time consuming when used to obtain a linear format.

Before applying our curve approximation method, we perform a small pre-processing on the border region. We “thin out” the border region by representing neighboring points with one representative point. More specifically we define a virtual sphere V, detect all the border regions points located inside V (suppose n is the number of points inside V) and calculate their center of mass, as shown in eq 4.1.

$$c_{x_i} = \frac{\sum_{i=0}^{n} x_i}{n}, \qquad c_{y_i} = \frac{\sum_{i=0}^{n} y_i}{n}, \qquad c_{x_i} = \frac{\sum_{i=0}^{n} z_i}{n} \qquad \text{(Eq. 4.1)}$$

We then find the border region point inside the sphere that is closest to the computed center of mass. This point is used as a representative for the group of points inside *V*. The virtual sphere is shifted to a new position (Figure 4.1(b)) and the process is repeated until we have obtained a thinner version of the point cloud. The size of the sphere determines the density of the thinned out cloud; the smaller the sphere, the denser the acquired point cloud. This method of thinning also provides us with an ordered sequence of points.

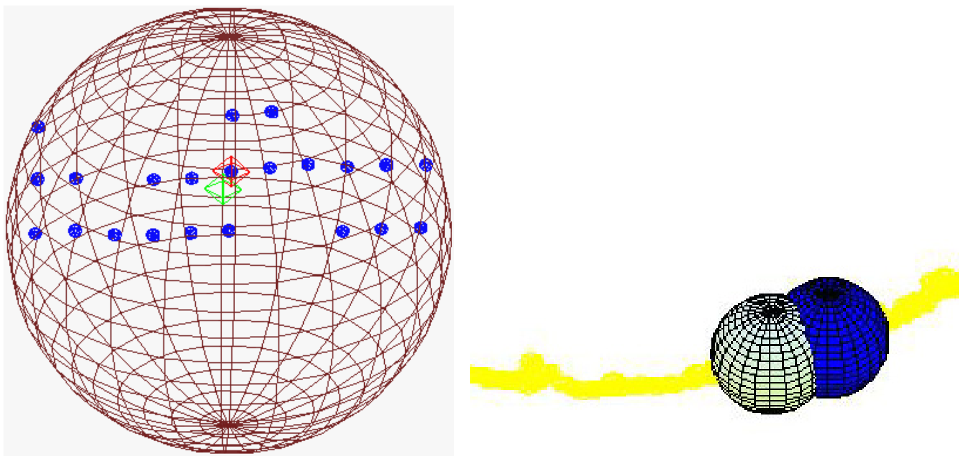


Figure 4.1 Virtual sphere containing border points. The green diamond corresponds to the center of mass, whereas the red diamond is the border point closes to it.

The sequence of points representing the feature border is divided into subsets of points and curve approximation is carried out on each curve segment.

Figure 4.2 A sequence of points to be approximated

Curve approximation is carried out with a least squares optimization procedure. Suppose $Q=\{Q_1,Q_2,\ldots Q_m\}$ (Figure 4.2) is a set of ordered border points and C is an approximating rational Bezier curve given by the equation:

$$C(u_i) = \frac{\sum_{j=1}^{n} w_j P_j B_j(u_i)}{\sum_{j=1}^{n} w_j B_j(u_i)} \qquad \text{(Eq. 4.2)}$$

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:

$$e_i = Q_i - C(u_i), \quad i=1..\text{m}. \qquad \text{(Eq. 4.3)}$$

We need to assign parameter values u_i to each point Q_i. We use chordal parameterization [44] in which we express the parameter value of each point Q_i in reference to its position in the point sequence:

$$u_i = \frac{\sum_{j=2}^{i} \Delta Q_j}{\sum_{j=2}^{m} \Delta Q_j} \qquad \text{(Eq. 4.4)}$$

The least squares problem is then to minimize the error:

$$E = \sum_{i=1}^{m} e_i^2 \quad \rightarrow \quad E = \sum_{i=1}^{m} (Q_i - C(u_i))^2 \qquad \text{(Eq. 4.5)} \quad \text{(Eq. 4.6)}$$

We consider the product w_jP_j as one variable and partially differentiate (Eq. 4.6 by factor w_kP_k, k=1..4. This leads to equations (for k=1..4):

$$\frac{\partial E}{\partial(w_k P_k)} = 0 \quad \Rightarrow \sum_{i=1}^{m} \left[2\Big(\sum_{j=1}^{4} w_j P_j B_j(u_i) - Q_i \sum_{j=1}^{4} w_j B_j(u_i)\Big) B_k(u_i) \right] = 0$$

(Eq. 4.7)

from which we obtain the following linear system of equations:

$$\begin{aligned} [B]^T [B][wP_x] &= [B]^T [Q_x B][w] \\ [B]^T [B][wP_y] &= [B]^T [Q_y B][w] \\ [B]^T [B][wP_z] &= [B]^T [Q_z B][w] \end{aligned}$$

(Eq. 4.8)

where (i.e. for coordinate x):

$$B = \begin{bmatrix} B_1(u_1) & B_2(u_1) & B_3(u_1) & B_4(u_1) \\ \vdots & & & \vdots \\ B_1(u_m) & \dots & \dots & B_4(u_m) \end{bmatrix} \quad wP_x = \begin{bmatrix} w_1 P_{x_1} \\ w_2 P_{x_2} \\ w_3 P_{x_3} \\ w_4 P_{x_4} \end{bmatrix}$$

$$Q_x B = \begin{bmatrix} Q_1 B_1(u_1) & Q_1 B_2(u_1) & Q_1 B_3(u_1) & Q_1 B_4(u_1) \\ Q_2 B_1(u_2) & & & \\ \vdots & \vdots & \vdots & \vdots \\ Q_m B_1(u_m) & \dots & \dots & Q_m B_4(u_m) \end{bmatrix}$$

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 reparametrizing u, but we need reparametrization 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.

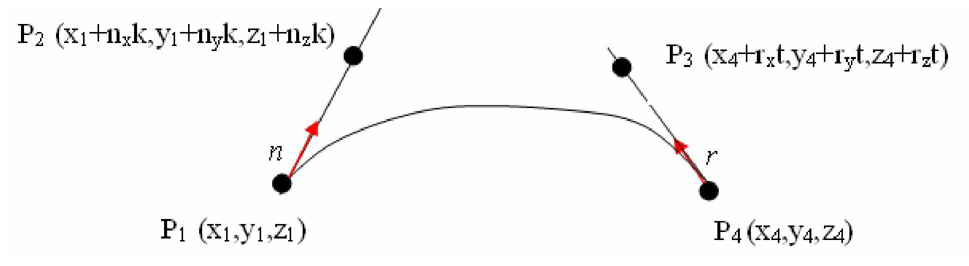


Figure 4.3 Inner control point coordinates expressed in reference to the end control points

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 [68].

Vectors *wPx*, *wPy*, *wPz* are modified by expressing the coordinates of inner control points P$_2$ and P$_3$ in relation to the end points. In eqn. 4.9 we now have:

$$\left[B^T\right][B]\begin{pmatrix} w_1(x_1+0) \\ w_2(x_1+n_x k) \\ w_3(x_4+r_x t) \\ w_4(x_4+0) \end{pmatrix} = \left[B^T\right][Q_x B][w] \qquad \text{(Eq. 4.9)}$$

Since we assume that w$_2$=1, the system is transformed so that its final form is (e.g. for direction x):

$$\left[B^T\right][B]\begin{pmatrix} w_1 x_1 \\ n_x k \\ w_3 r_x t \\ w_4 x_4 \end{pmatrix} = \left[B^T\right][Q_x B][w] - \left[B^T\right][B][A_x] \quad , \quad [A_x] = \begin{bmatrix} 0 \\ x_1 \\ w_3 x_4 \\ 0 \end{bmatrix} \qquad \text{(Eq. 4.10)}$$

These systems of linear equations (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 eq. 4.5 as follows:

$$e_i = \sum_{i=1}^{m} (e_{x_i}^2 + e_{y_i}^2 + e_{z_i}^2) \qquad \text{(Eq. 4.11)}$$

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:

$$\begin{aligned}&\sum_{i=1}^{m}\left(B_k(u_i)\left(Q_{x_i} Q_{x_i} \sum_{j=1}^{4} w_j B_j(u_i) + Q_{y_i} Q_{y_i} \sum_{j=1}^{4} w_j B_j(u_i) + Q_{z_i} Q_{z_i} \sum_{j=1}^{4} w_j B_j(u_i) \right)\right) = \\ &\sum_{i=1}^{m}\left(B_k(u_i)\left(Q_{x_i} \sum_{j=1}^{4} w_j P_{x_i} B_j(u_i) + Q_{y_i} \sum_{j=1}^{4} w_j P_{y_i} B_j(u_i) + Q_{z_i} \sum_{j=1}^{4} w_j P_{z_i} B_j(u_i) \right)\right)\end{aligned} \qquad \text{(Eq. 4.12)}$$

Eventually we end up with the linear system:

$$\begin{aligned}&\left([Q'_x B]^T [Q'_x B] + [Q'_y B]^T [Q'_y B] + [Q'_z B]^T [Q'_z B] \right) [w'] = \\ &[Q'_x B]^T [B][w' P'_x] + [Q'_y B]^T [B][w' P'_y] + [Q'_z B]^T [B][w' P'_z] + (-1)\left(\left([Q'_x B]^T [C_x] \right) + \left([Q'_y B]^T [C_y] \right) + \left([Q'_z B]^T [C_z] \right) \right)\end{aligned} \qquad \text{(Eq. 4.13)}$$

where (e.g. for x coordinate):

$$Q'_x B = \begin{bmatrix} Q_1 B_1(u_1) & \cdots & Q_1 B_4(u_1) \\ \vdots & \ddots & \vdots \\ Q_m B_1(u_m) & \cdots & Q_m B_4(u_m) \end{bmatrix} \qquad Cx = [Q'_x B] \begin{bmatrix} P_{x_2} B_2(u_i) \\ \vdots \\ P_{x_2} B_2(u_m) \end{bmatrix}.$$

This procedure is carried out iteratively until the error function is minimized. An example of the result of this method is shown in Figure 4.4. This method reaches the best solution after 10 iterations. The red points correspond to points of the curve segment whereas the black points are the respective points calculated on the rational Bezier curve.

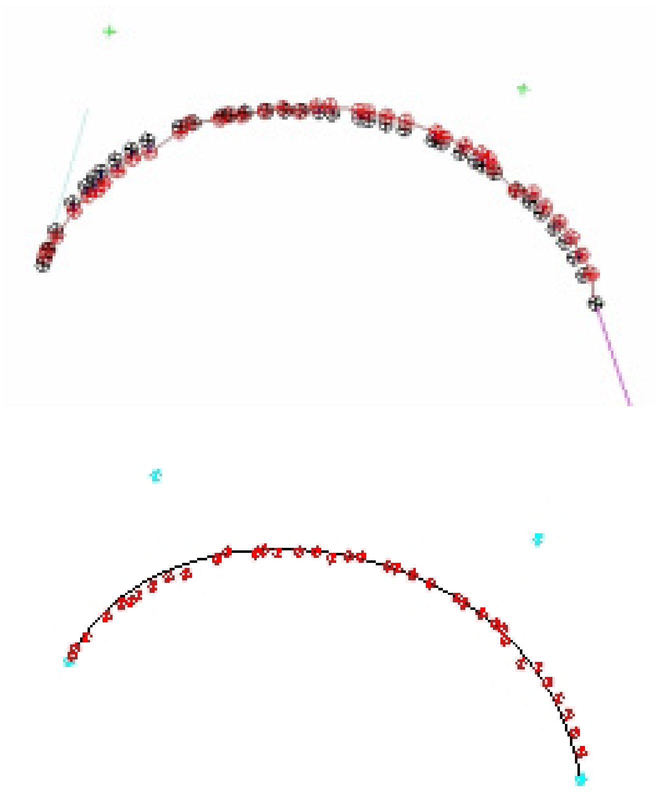


Figure 4.4 (top) Points used for approximation (red) and corresponding points computed on curve (black), (bottom) Cubic rational Bezier curve approximating the sequence points (red) and the corresponding control points (turquoise).

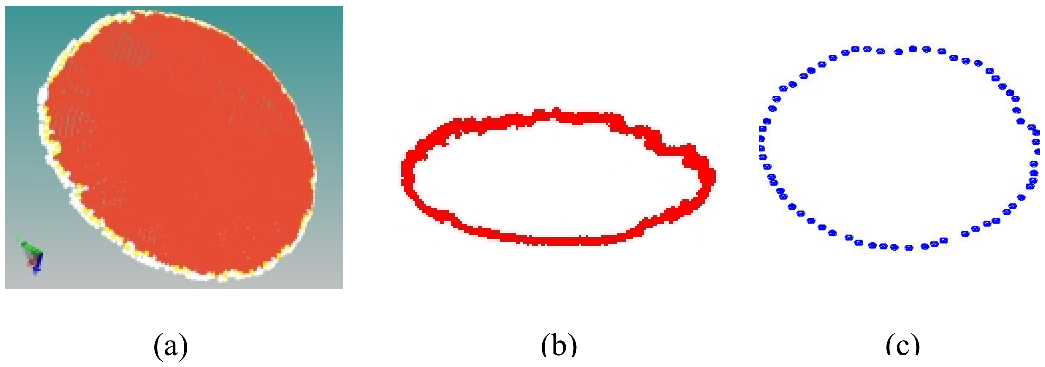
Generally this is a fast curve approximation approach that produces smooth continuous curves that interpolate or pass close by the data points. A good approximation is reached within a few iterations. In some cases, the minimal error is reached after the first iteration, if the point cloud data is not 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.

4.4. Examples of curve fitting using rational Bezier curves

In this section we will present examples of our curve fitting method which was implemented using Maple 11. The data sets used in the examples are areas corresponding to feature boundaries returned by our region growing algorithm. For each point in the border point cloud we have computed in a previous phase (feature detection phase) its corresponding surface normal vector estimate. The boundary point set is divided into subsets (curve segments) based on the progressive change in the tangent vector. The point cloud is divided into as many sets needed, so that the angle formed by the tangent vectors of the start and end point of each section is below a threshold.

4.4.1. Example A

Figure 4.5 shows one of the region borders detected by our region growing method. This border contains 450 3D points. We perform thinning on the border region and we estimate the tangent vector at the each point using the nearest neighbors. Then we divide the thinner version of the border into segments by comparing the starting tangent vector of the segment with the sequence of points, until the angle formed by the vectors is larger than a threshold angle value. In the following example, the border region is divided into 3 segments. If a smaller threshold is used, then more curve segments will be obtained.



(a) (b) (c)

Figure 4.5 (a) Region detected by region growing, (b) the border of the region to be approximated, (c) a thinned-out border

We apply the curve fitting algorithm to each curve segment. The final curve approximating the point cloud is shown in Fig. 4.6. The first set of points used for approximation consists of m=40 points, with the distance between the end points corresponding to 0.0279 and the average error is given by $E/m=0.164\cdot10^{-6}$. For the second approximating curve, the set of points used consists of m=36 points with end points distanced at 0.024 and average error is $0.708\cdot10^{-6}$ after one iteration. The final segment consists of m=34, the distance between the end points is 0.0229 and, after one iteration, average error is $0.825\cdot10^{-6}$. The results of the curve approximation are summarized in table 4.1.

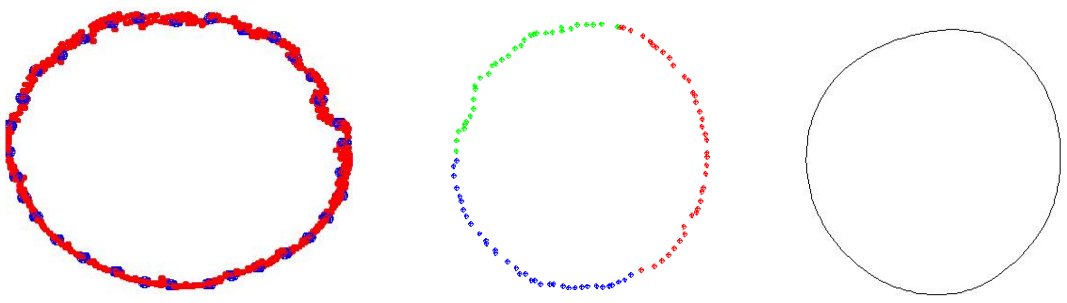


Figure 4.6 (a) The thinned out border region (b) the curve segments to be approximated (#1-green, #2=blue, #3=red), (c) the final approximating curve

Curve segment	# of points	Distance between end points	Average error E/m	# of iterations
#1	40	0.0279	$0.164 \cdot 10^{-6}$	4
#2	34	0.0229	$0.825 \cdot 10^{-6}$	1
#3	26	0.024	$0.708 \cdot 10^{-6}$	1

Table 4.1 Results of Curve Approximation

4.4.2. Example B

Fig. 4.7 shows another region detected with the region growing algorithm when applied to the screwdriver point cloud. The border of this region consists of 498 points which is divided into three segments as show in figure 4.7(c). After thinning, the tangents at each end point are estimated and the curve approximation method is applied to each segment. The resulting curves are shown in fig. 4(d). The first segment consists of m=33 points, with the Euclidean distance between the end points corresponding approximately to 0.02. After 10 iterations the least squares error function is minimized and the ratio average error E/m is $0.64 \cdot 10^{-6}$. The second segment consists of m=19 points, the distance between the end points is 0.014 and the average error is $0.14 \cdot 10^{-6}$ after 11 iterations (the sum of squared errors is $0.27 \cdot 10^{-6}$). Finally, the third segment consists of m=27 points, the distance between the end points is 0.017 and after 30 iterations the average error is $0.46 \cdot 10^{-6}$. These results are summarized in Table 4.2.

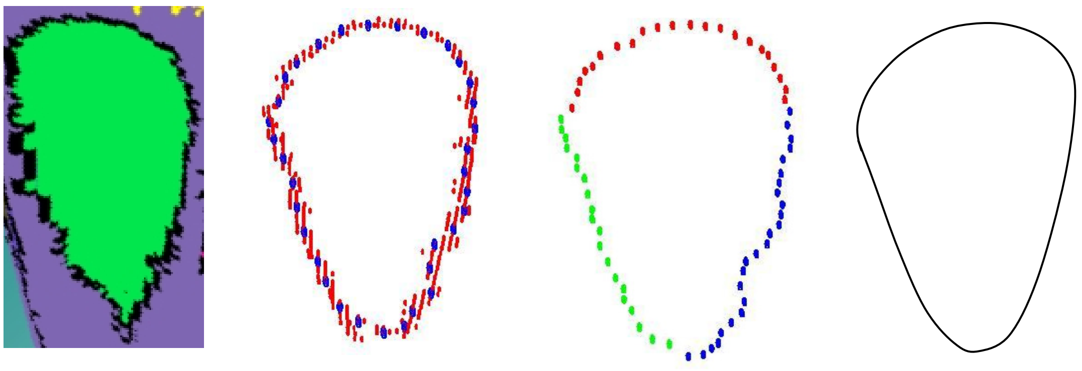


Figure 4.7 . (a) The region created by region growing (b) the border of the region, (c) the sample of points used for approximation, (d) the final curve approximation of the border.

Curve segment	# of points	Distance between end points	Average error E/m	# of iterations
#1 (green)	33	0.02	$0.64 \cdot 10^{-6}$	10
#2 (red)	19	0.014	$0.14 \cdot 10^{-6}$	11
#3 (blue)	27	0.017	$0.46 \cdot 10^{-6}$	30

Table 4.2 Results of Curve Approximation

4.5. Evaluation of our method

To evaluate the result of our approximation method we performed an optimization to compute the absolute optimal curve using the IpOpt software [47]. We perform optimization treating u_i, i=1..n, the weights and the control points as unknowns and we minimize the objective function of eqn. 4.5 under the following constraints:

$$(Q_i - C(u_i)) \cdot C'(u_i) = 0 \qquad \text{Eq. 4.14}$$

where $0 \le t_i \le 1$ and i=1..n. After performing various experiments we observed that our approach provides satisfactory results in significantly less time. In one case, for example, the sum of the squared errors after convergence was $1.279 \cdot 10^{-6}$ after approximately 1000 iterations as opposed to $2.4 \cdot 10^{-6}$ after 12 iterations as provided by our approach. In general, from repeated experiments, we have determined that the average error of the curve computed by our approach is about 0.5-2.5 times larger than the average error of the optimal curve. Also the average error computed by our method is larger than the actual error for the curve derived by our approach. Therefore we can conclude that the curve constructed by our method is a satisfactory approximation.

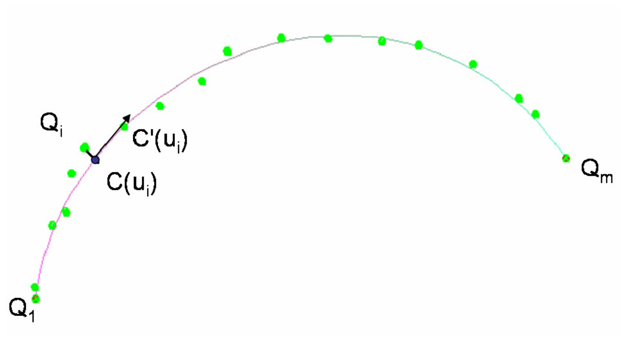


Figure 4.8 A sequence of points approximated by a curve C, under the constraints of Eq.4.14

In general, we have developed a fast curve fitting method which derives cubic rational Bezier curves conforming to G^1 continuity constraints. By imposing continuity constraints into the least squares optimization process we ensure that the computed control points respect the estimated tangents at the end points. We also adjust and control points and weights until the error of the least squares is minimized. The weights are used not only to affect the shape of the curve but also to adjust the parametrization of the curve.

As compared to other curve approximation methods which perform curvature-based fairness our method is effective and efficient with satisfactory accuracy. This is important given that the application in which this method is employed calls for the reconstruction of the borders of all features, which in the case of freeform objects, may be a few hundreds and quite complex.

CHAPTER 5. FEATURE RECONSTRUCTION

In this chapter we consider how feature components that consist of feature regions and boundary curves can be used to reconstruct the complete CAD model of an object. We apply solid modeling techniques to combine and obtain the entire 3D model representation. Furthermore we introduce parameters and constraints on these features to allow editing and capture design intent.

5.1. Features in reverse engineering

"Feature", in computer-aided design, is a new term that is used to describe an entire class of concepts. In general, features are generic shapes or characteristics which can be associated with certain attributes and knowledge [71]. These attributes can describe the morphology of the object, i.e. through parameters defining its size, shape and orientation, and its behavior in a CAD model, i.e. connectivity issues and constraints. The use of features in a model provides the user-designer with the ability to edit and redesign the model [41]. A user-designer can even define his own feature components to use in the design process [40]. Given this, various applications apply the concept of features in a way compatible to the scope and aim of the respective application.

In reverse engineering applications, and specifically, in reconstructing mechanical parts, features are shapes traditionally used in engineering designs such as slots, holes, and bosses [85]. In reverse engineering objects of more freeform design, the term "feature" is often used in the sense of feature lines [27], meaning edges/boundaries of an object that can be detected by changes in the surface curvature of the object [25],[39]. [88] use an intermediate data structure called a feature skeleton which is a network of curves basically representing the boundaries of region sets. In Brep models features can also be surfaces that can be grouped together based on certain characteristics or properties. In this sense, we consider as a feature a group of points that has a morphological meaning, from a design point of view. For example, in a point cloud of a duck, a point set corresponding to the head can be considered a feature. In a screwdriver, the points corresponding to the handle may be considered a feature. But features can be even more specific. For example, the duck's head can be divided into more features i.e. beak and the rest of the head (Figure 5.1).

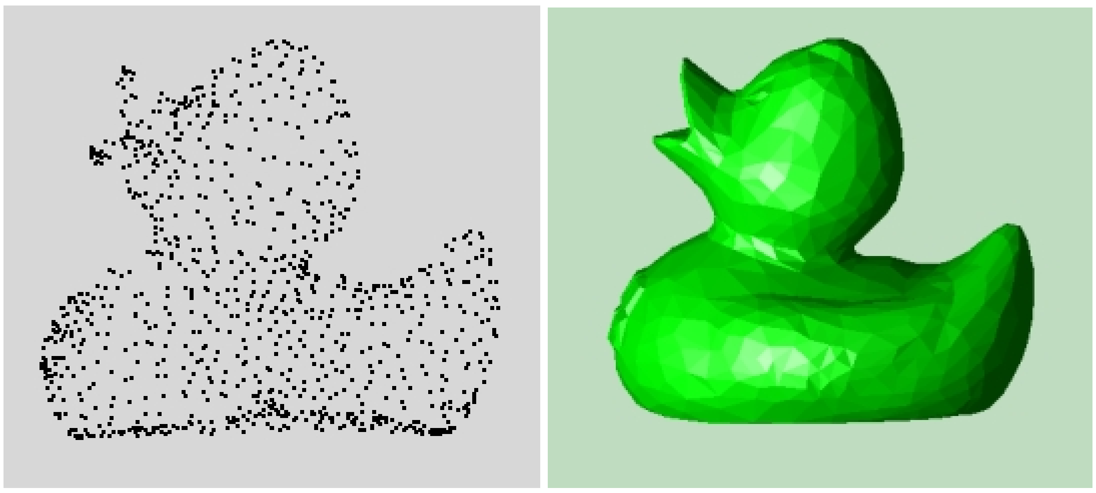


Figure 5.1 (left) A point cloud of a duck (right) a mesh representation of the duck

5.2. Solid modeling techniques

Features, as used from our perspective, can be reconstructed from their regions either by surface fitting or solid modeling functions. Surface fitting would provide us with a reconstructed surface model that would conform to the initial object from both a morphological and aesthetic point of view, however its editing capabilities would be limited. Therefore we choose to apply solid modeling techniques to our feature regions so that the reconstructed features are more easily edited and modified. The modeling techniques we focus on are sweeping, skinning, covering and blending.

Sweeping is a modeling function in which a closed planar domain is translated (translational sweeping) along a trajectory curve (figure 5.2) or rotated (rotational sweeping or swinging) around an axis to form a solid. If an open domain is swept accordingly then a surface model is formed. [55]. Work on reverse engineering point clouds using sweeping techniques has been done by [49]. The author performs slicing on a point cloud using the bounding box as a guide and reconstructs a boundary curve conforming to the points retrieved from slicing. The boundary curve is swept accordingly to reconstruct and obtain the CAD model.

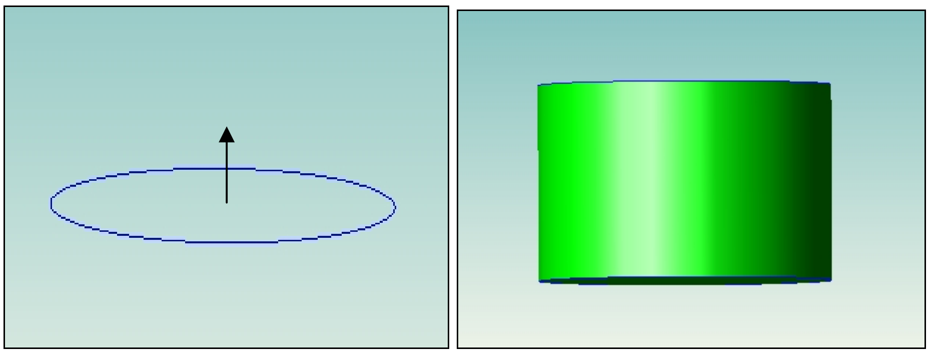


Figure 5.2 An example of translational sweeping

Skinning is a modeling function where a closed volume or solid is formed by creating a skin surface over prespecified cross-sectional planar surfaces (figure 5.3) whereas covering is a function that covers (that is, fits a surface onto) closed boundary curves in solid or wireframe objects (figure 5.4).

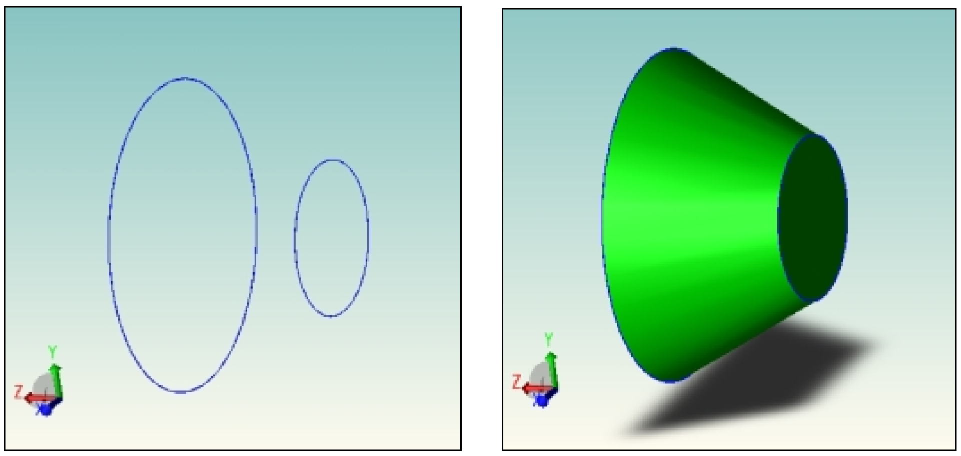


Figure 5.3 Example of skinning between two parallel circular curves to create truncated cone

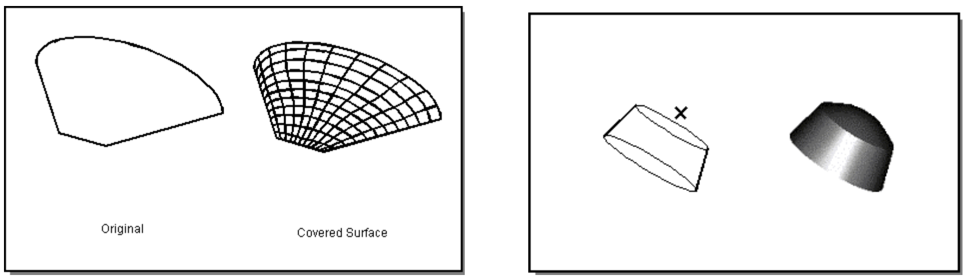


Figure 5.4 (left) Covering a boundary with a face, (right) adding a point to a solid by covering.

Blending is a function used to modify a model so that a sharp edge or vertex is replaced by a smooth surface whose normal vectors are continuous with those of the surfaces that originally meet at the edge or vertex. This function is used mainly for aesthetic reasons, to make the model look smoother.

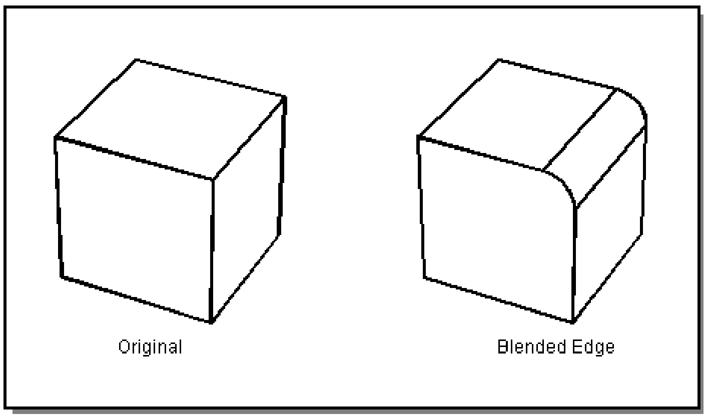


Figure 5.5 An example of blending

Covering, skinning and blending are techniques where surfaces are fitted accordingly to boundary curves or other primitives. The type of modeling function used during the reconstruction process depends on the types of curves and symmetries detected in our feature regions, as is described in the following section.

5.3. Generating a feature assembly plan

We consider how features can be reconstructed from the subsets of points produced by our point cloud segmentation approach using solid modeling techniques to capture the design intent and semantics of the model. Our main focus is to reconstruct each feature as a solid entity, preserving the shape morphology and semantics, without necessarily interpolating the point cloud exactly. We apply symmetry detection methods to determine what modeling technique is more appropriate in each case.

Initially we calculate the oriented bounding box (OBB) that is aligned accordingly with the feature region point cloud [38]. This is carried out by using principal component analysis (PCA) [29] [49] to find the axes of the oriented bounding box. The dimensions of the OBB axes are determined by projecting the point cloud onto each axis and using the extreme values as the axis dimension values. The PCA provides the most significant axes of the point cloud and basically expresses the way the point cloud is distributed by looking at the covariance of the points. To compute the PCA, first we find the average position $M(m_x,m_y,m_z)$ (center of mass) of all the points of the feature region and then we compute the covariance matrix of the point cloud. The covariance matrix for every point P of the feature region is expressed by eq. 5.1 and is a symmetric 3×3 matrix. The eigenvectors of the covariance matrix give the directions along which the point cloud is most and lest spread out.

$$C = \frac{1}{n}\sum_{i=1}^{n}(P_i - M)(P_i - M)^T \qquad \text{(Eq. 5.1)}$$

By this method we derive the principal axis of the feature point cloud and the OBB. The next step is to determine for each feature region which solid modeling technique is more appropriate for reconstruction. This depends heavily on the distribution and the geometry of the point cloud.

5.3.1. Sweeping

If the point cloud is distributed distinctly along one of the principal axes forming i.e. a long feature, such as the screwdriver shaft, then reconstruction is initiated by slicing the feature point cloud. A slicing plane is moved along the main principal axis of the OBB and cross sections of the point cloud are obtained. Points of the point cloud not on but close to the plane are projected onto it. The points derived from slicing are used in our contour reconstruction method (as a 2D curve approximation problem) for curve fitting. The slicing plane *S* is moved along the main principal axis, used as the slicing path, until the length of the bounding box is traversed. Sequential cross-sections of the feature point cloud derived by this method are compared to determine if any similarities or symmetries exist and in what form. Reconstructed cross-section curves with that can be considered identical imply the application of sweeping for reconstruction.

For example, cross-sections of the upper half of the screwdriver shaft result in round circular curves that, when compared, match. This implies that reconstruction at that part of the feature should be carried out with translational sweeping. Before carrying out reconstruction, we examine all sequential cross-sections to define the limits of the sweeping function. Sweeping is performed on the trajectory up to the point where the other end of the OBB is reached or the cross-section contour changes. In the example of the screwdriver shaft, this occurs up to the point where the tip of the shaft starts to form. This type of sweeping, along a linear trajectory path, is also referred to as extruding.

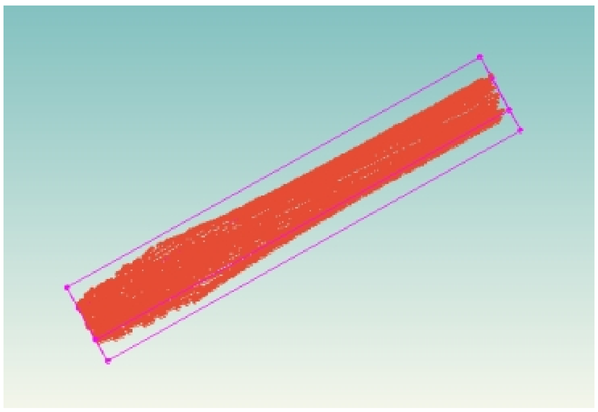


Figure 5.6 Feature region corresponding to a screwdriver shaft

5.3.2. Skinning

In the case where two cross-section curves are different, in size or shape, reconstruction is performed through skinning techniques. In the previous example of the shaft reconstruction, from the point where the sweep function ends, up to the tip of the shaft, where we reach a feature boundary curve, sequential cross-sections cannot be matched, thus leading to the application of skinning techniques to create surface patches between the two cross-section curves. In general, if a network of different cross-section curves is provided, then skinning is performed using the intermediate curves as guides for the skinning operation.

In some cases features can be reconstructed in more than one ways. Let us consider for example the bottom surface of the screwdriver that is connected with screwdriver shaft (figure 5.7). This feature has 2 border curves, the outer and the inner boundary. This feature can easily be reconstructed using skinning between the two boundary curves. However it can also be reconstructed using rotational sweeping. By creating the OBB of the feature region we also obtain the principal axes. The axis that passes through the hole can be used as an axis of rotation for sweeping. The profile curve for sweeping is obtained by slicing the point cloud with a plane that originates from the rotation axis and is parallel to one of the faces of the bounding box. The points on and very near to the slicing plane are used for the profile curve reconstruction, which is then swept around the axis accordingly.

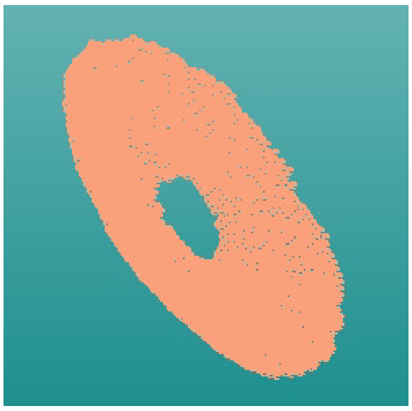


Figure 5.7 A feature representing the bottom of the screwdriver handle

Reconstructing features using sweeping and skinning is summarized in Table 5.1

```
Feature Reconstruction with sweeping and skinning
1. Compute the PCA of the feature region
2. Find OBB based on principal axes
3. Create slicing plane S parallel to OBB and perpendicular to selected PA
4. Until path $T$ (length of OBB axis) is traversed
    4.1. Move S on T
    4.2.Find all points $P_i$ of feature region located on or close to $S$
    4.3 Apply curve approximation on $S$ to construct curve $c$
5. Detect similarities/characteristics of curves
6 Apply appropriate modeling function for feature reconstruction
    6.1 If cross sections are identical $\rightarrow$ perform translational sweeping
    6.2 If cross section differ $\rightarrow$ perform skinning
```

Table 5.1 Reconstructing feature regions with sweeping / skinning

5.3.3. Covering

Some feature regions can easily be reconstructed using covering techniques. This holds for point cloud regions that are fairly flat, spread out and/or freeform. Suppose we have the feature region shown in Figure 5.8 and its corresponding boundary box.

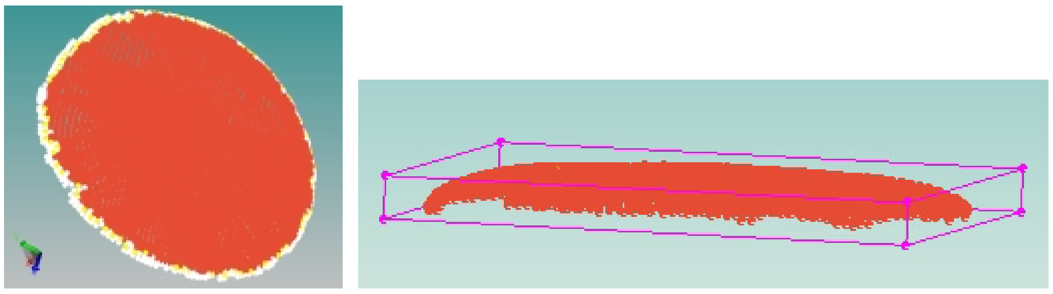


Figure 5.8 Top part of screwdriver handle and its corresponding boundary box

The specific feature point cloud is very thin and spread out, similar to an overturned plate. Using the feature's boundary curve and a sample of points located on the top of the feature (on and very close to the upper face of the OBB) we create a surface that interpolates the points providing a smooth result. In general, covering techniques can be applied to cover points or other guide contours.

5.3.4. Assembling the features

A point cloud is reconstructed based on a feature assembly plan. A feature assembly plan provides a logical structure for determining the feature parts to be reconstructed and how they are connected. The feature assembly plan of a point cloud can be derived based on a decomposition of an object into its features. We consider the screwdriver point cloud as our example case. We construct a feature connectivity graph G(v,e) where v are the nodes of the graph and e are the edges. Every node of the graph corresponds to a feature in the point cloud, whereas an edge between two nodes means that the two features are connected. An initial, if not final, estimation of the feature connectivity graph is provided by our region growing feature detection and extraction algorithm.

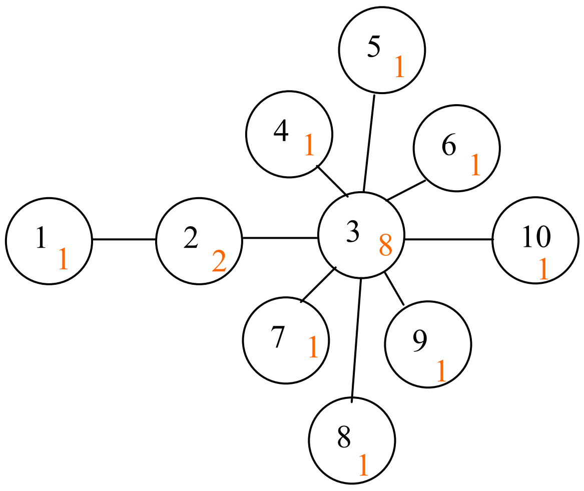


Figure 5.9 Feature Connectivity graph of screwdriver

A feature decomposition tree (Figure 5.10) of the screwdriver is derived from the feature connectivity graph. The screwdriver's features can be decomposed into the following parts:

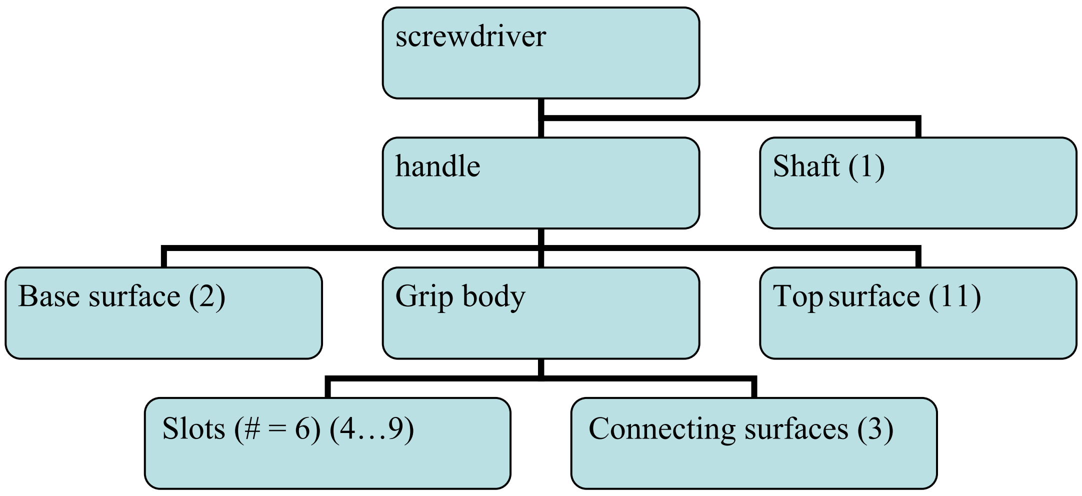


Figure 5.10 Decomposition of the screwdriver model into its feature elements

The connectivity graph is augmented with data structures for each feature node containing information relevant to their corresponding features, such as number of points, average normal vector, average intensity value. The edges of the graph also have corresponding data structures describing information such as the number of points in the border, the connected features etc.

The second number (in red) in every node describes the degree of the node. From the graph in Figure 5.9 we determine that all the nodes except for node 3 have a low degree of 1 or 2. Node 3 however has a degree of 8. Also nodes 1, 10 and 4-9 have one boundary contour each, whereas node 2 has two boundary contours and node 3 has eight boundary contours. Node 3 has two boundary connections with nodes 2 and 10 and 6 distinct border regions around the slots (one for each slot).

If a node in the connectivity graph is of high degree or has a large number of boundary contours, such as node 3 in Figure 5.9, then further decomposition is performed on the feature corresponding to the node. For example, the grip body of the screwdriver can be decomposed into smaller regions of a reduced degree with less boundary contours (Figure 5.11). This results in more feasible solid modeling steps for the reconstruction of the point cloud.

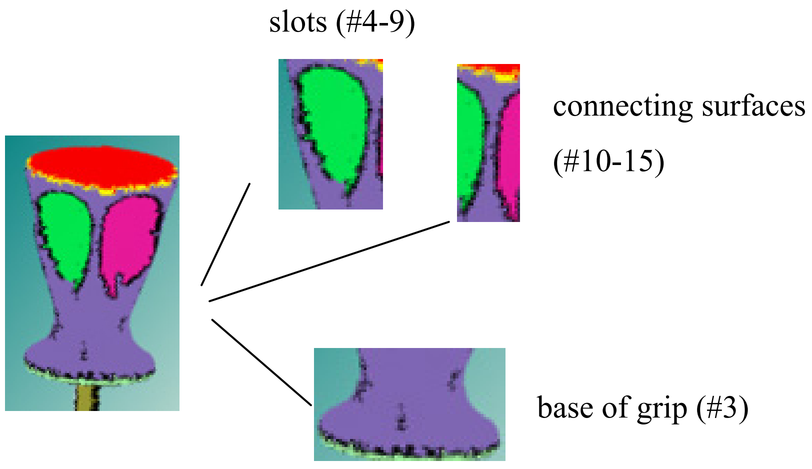


Figure 5.11 Decomposition of the grip body of the handle

Specifically, the grip body is divided into the base of the grip, the six slots and six intermediate connecting surfaces which surround the slots. The corresponding feature connectivity graph is as follows:

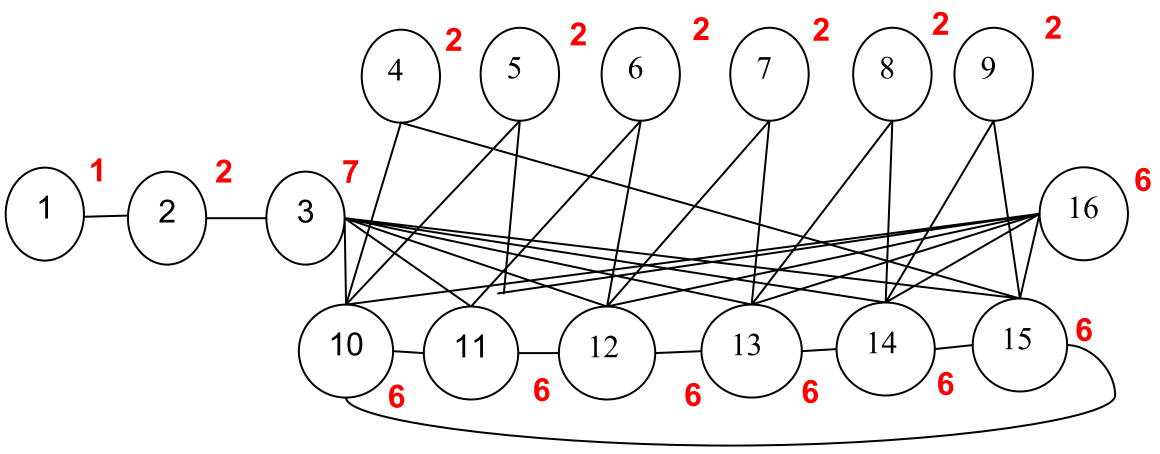


Figure 5.12 The feature connectivity graph after decomposition of node #3

Each feature entity in this graph contains one or two at the most continuous border contours. Specifically, the nodes 1, 4-9, 10-15 and 16 have one boundary contour each, whereas nodes 2 and 3 have 2 boundary contours. Also the highest degree of a node is seven, in contrast to the previous graph where the highest degree was eight. Therefore, the highest degree of the nodes and the number of boundary contours per node has been reduced.

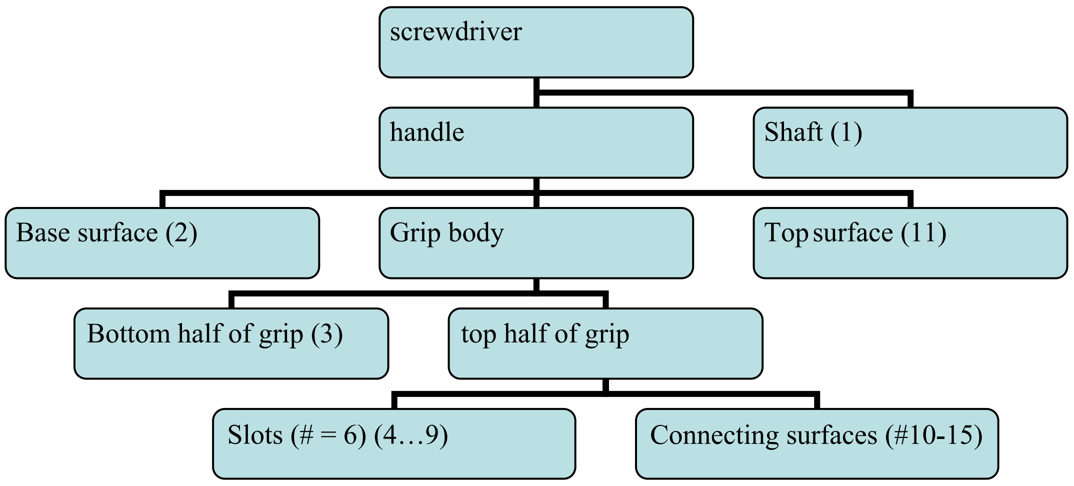


Figure 5.13 Feature decomposition tree of the screwdriver after further decomposition of the handle grip body

5.4. Parameter definition and editability

The major advantage of feature based modeling is that by defining parameters on the features, the user-designer can more easily and efficiently modify the specific feature without necessarily affecting the rest of the solid model. While a feature is a physical entity that makes up the physical part, an attribute (property) is a characteristic or a quality of a feature [71]. The parameters (attributes) defined on a feature may refer to a number of things. Some attributes may refer to the geometry or topology of the feature shape, its dimensions, location, orientation and construction method. Attributes may even describe properties such as material or texture of the feature. Usually parameter definition is accompanied by the definition of constraints to which the model must conform. We look to define parameters and constraints on our features that determine the main functionality and behaviour of the features in the model, i.e. dimension limitations, basic connectivity, to provide basic editing capabilities.

5.4.1. Parameter definition

The parameters and constraints we define on our reconstructed features refer to the:
Position and alignment in the point cloud
Neighbour connectivity
Construction method

Position and alignment

The position of all the features in a point cloud A can primarily be defined in reference to the center of mass of the whole point cloud cm(A). The OBB of a point cloud A is calculated, providing the principal axes and the center of mass cm(A). For each feature f_i in the point cloud that is reconstructed we define:

- the distance of the center of mass of f_i from cm(A), $d(\mathrm{cm}(f_i),\mathrm{cm}(A))$

- the alignment of the feature in reference to A. We compute the angle formed by the normal vectors of the planes passing through the middle of the bounding boxes that contain the principal axes.

- the size of the feature can be defined by the size of the OBB

Neighbour connectivity

For each feature we define parameters that refer to the neighbors of the feature and their point of connectivity. Specifically for each feature *i* and its neighbor f_j we define

- the distance d(cm(f_i), cm(f_j))
- the boundary curve of f_i that connects with neighbor f_j

Construction Method

Each solid modeling technique requires its own set of parameters that define how the function is performed. Specifically we define:

- Translational sweeping: profile curve and the trajectory curve.
- Rotational sweeping: profile curve and the axis of rotation.
- Skinning: network of cross-section curves used for skinning
- Covering: boundary curve and any guide primitives used (i.e. points, other curves)

5.4.2. Editability

By defining parameters on our features, we provide the capability of editing the feature. Changing a parameter of a feature means that the model has to be re-evaluated and reconstructed to conform to the new parameter values. Depending on the parameter that is changed, the new feature model may be slightly or completely different from its previous state. For example, if one of the intermediate curves used in skinning a feature is made larger, then the resulting model will be different from its previous instant. However, if the guide curve is changed dramatically, i.e. the shape of the curve changes, then the edited model will be very different.

Editing functions are closely linked to constraint definition. Most often, parameter values should be constrained with upper and/or lower bounds, so as to make sure that the model's design intent and functionality is not compromised. Suppose we change the length parameter value of the screwdriver shaft. If this parameter is left unconstrained, then a value may be provided that makes the shaft too long to be of any practical use. Therefore, in cases like this, the parameter values must be bounded.

Inter-feature constraints are defined to prevent inconsistent neighboring components. For example, the shaft of the screwdriver connects with the bottom surface of the handle. If the diameter of the shaft is modified (scaled up or down), then the boundary hole of the handle bottom has to be modified appropriately, so that the model is accurate. Also, inter-feature constraints such as parallelism and perpendicularity aid the reconstruction process by setting standards that may overcome noise problems or anomalies that may exist in the point cloud.

Intra-feature constraints are used to define and change the morphology of the feature. Constraints are defined on parameters such as the length between cross sections (i.e. for skinning), the dimension of the feature, the trajectory path used for skinning and sweeping and any other properties that contribute to the resulting constructed model. Also constraints can be applied to the points used to reconstruct contours to allow more difficult modifications, such as skewing.

5.5. Persistent naming

When editing parametric feature based models there are some issues that can emerge that concern the robustness and correctness of the modeling process and its result. An interesting problem that can arise during parametric modeling is that of persistent naming. Persistent naming concerns the characterization of geometric and topological entities of a parametric model so that they can be identified at any time of the redesign process. The entities of a parametric model may be modified and re-evaluated at any time in the design process. It is important that every modification or re-evaluation is performed on the correct instant of an entity. So persistent naming refers to naming the entities in such a way that reevaluation of the model leads to correct and consistent results. An insightful state of the art survey is provided by [60], which presents various approaches to the persistent naming problem. An example of the persistent naming problem is provided in figure 5.12. Specifically, an initial model is constructed by sweeping. A horizontal slot is created on F1 resulting in two feature region f2 and f3. A rounding function is applied to edge e of feature f3. When re-evaluating the model, a horizontal slot is created in such a way that edge e basically now corresponds to edges e1 and e2. Therefore, in some way, edge e has to be mapped to edges e1 and e2 so that rounding can be performed on both edges.

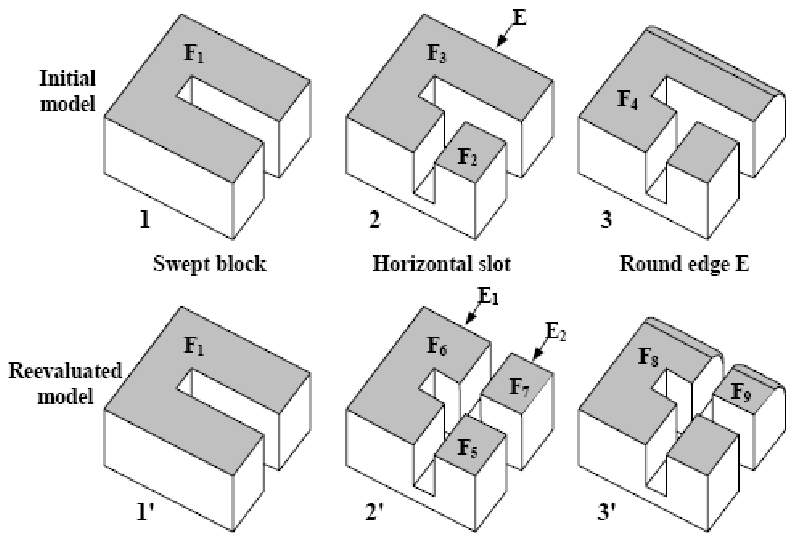


Figure 5.14 An example of the persistent naming problem [60].

Work on this problem has been done by [19, 41], who provide a scheme for storing feature information regarding topology and geometry (E-rep) and matching the information efficiently to deal with the naming issue. A mapping technique using topological ids for topological entities in a parametric model is suggested by [50]. [1] presents a naming scheme to identify entities used in parametric modeling by apprehending design intent. In [67] persistent naming is examined in reference to boundary representation deformation in solids.

CHAPTER 6. EXAMPLES

In this chapter we will present a few examples from the implementation of our different approaches.

6.1. Feature detection and extraction examples

In this section we will present examples of applying our feature detection and extraction algorithm on different point clouds [2, 12, 23, 77]. The algorithms for point concavity intensity computation and region growing have been implemented and tested under the Microsoft Visual C++ programming environment using ACIS R18 solid modeling libraries by Spatial. The GUI of the application has been implemented using HOOPS 16.20.

- Example A

We apply our feature detection and extraction algorithm on a point cloud of a monkey containing 2807 points. Region growing detects approximately 12 regions:

- Head body
- 2 regions for each ear (inner and outer areas)
- Brow area (2-3 regions)
- Eyes
- Nose
- Mouth-chin area

- Mouth

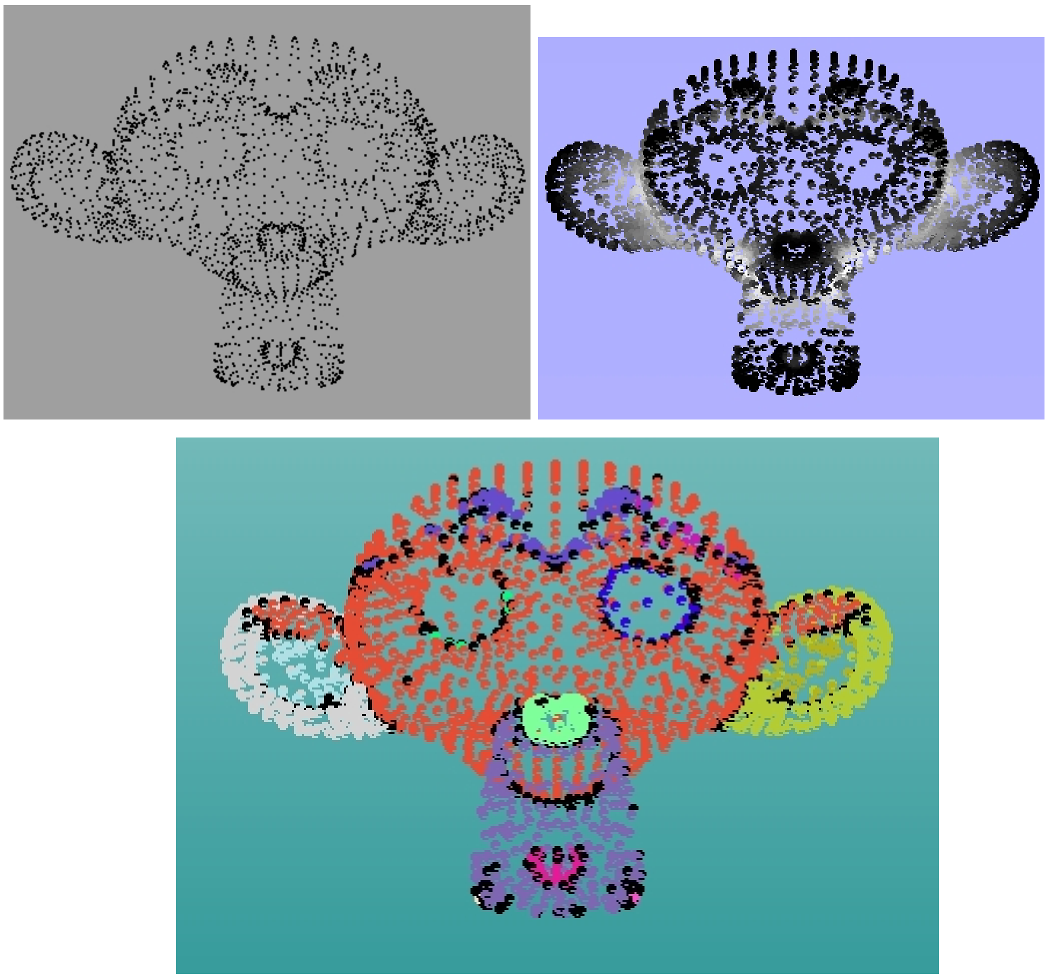


Figure 6.1 (top left) A point cloud of a monkey, (top right) the point concavity intensity map, (bottom) the features detected by region growing.

- Example B

Our algorithm is performed on a point cloud of a duck consisting of 965 points. Six regions are detected:

- Head
- Upper part of the beak,
- Lower part of the beak,
- Body,
- Surface under the body and

- a circular region on the model's bottom surface

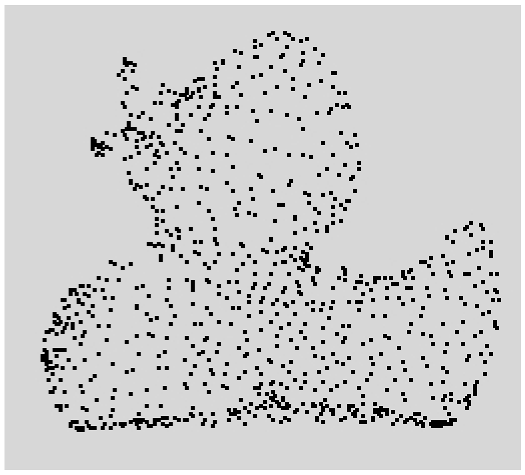
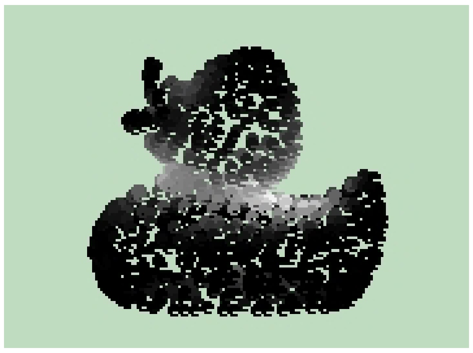
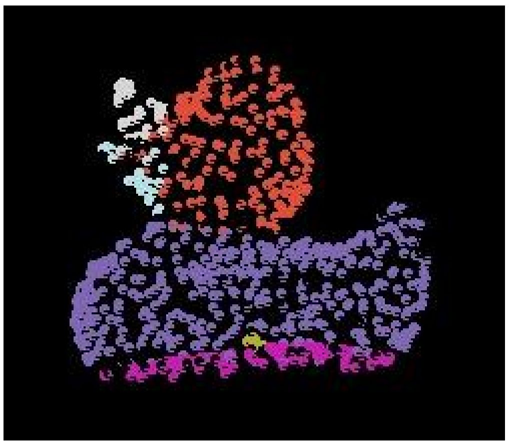




Figure 6.2 (top left) The point cloud of a duck, (top right) the concavity intensity map for the corresponding point cloud, (bottom) the regions detected by region growing.

6.2. Contour Reconstruction

The contour reconstruction approach is implemented and tested using Maple 11. This method has been tested in 2D and 3D.

-Example A

Suppose we have a cross-section from the screwdriver shaft derived by slicing the region with a plane. We reconstruct curves that best fit the points on the plane. For the means of this example, we choose a small angle for cutting the point cloud into curve segments and retrieve four curve segments. We perform approximation on each curve segment and retrieve the results summarized in table 6.1.

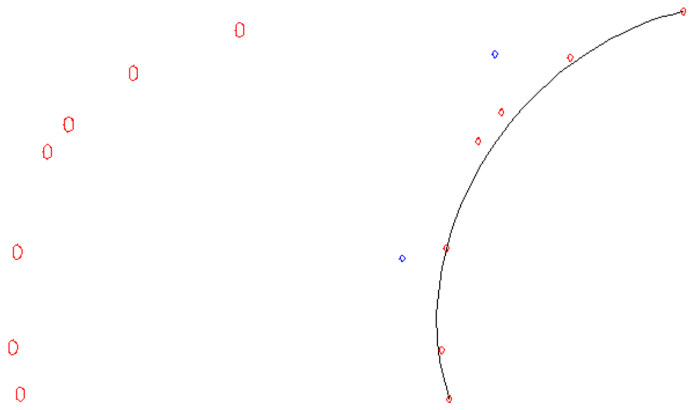


Figure 6.3 Curve approximation performed on the first curve segment

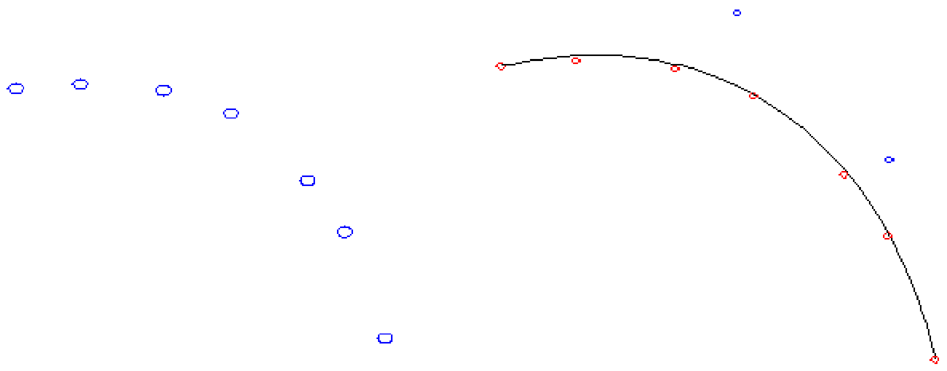


Figure 6.4 Curve approximation performed on the second curve segment

Curve segment	# of points	Distance between end points	Average error E/m	# of iterations
#1	7	0.0029348555	$8.573 \cdot 10^{-9}$	1
#2	7	0.002959652851	$2.336 \cdot 10^{-9}$	1
#3	6	0.002539316640	$8.618 \cdot 10^{-10}$	1
#4	6	0.003147043215	$1.1816 \cdot 10^{-8}$	1

Table 6.1 The results of contour reconstruction

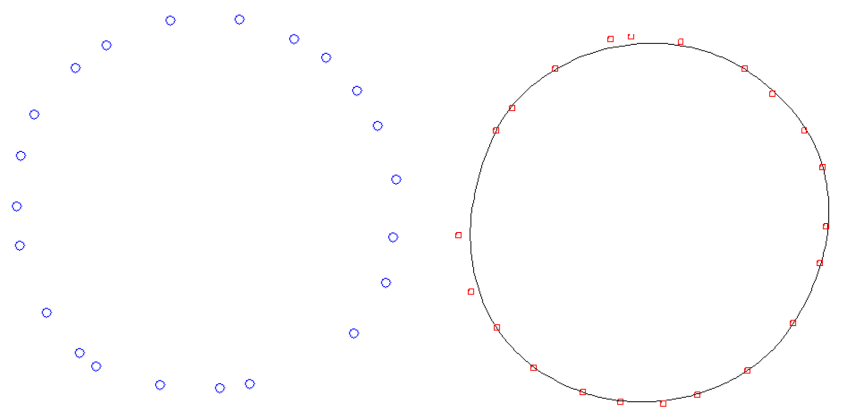


Figure 6.5 (left) The point cloud to be approximated (right) the reconstructed contour

From this table of results we observe that curve approximation works extremely well when the scanned data is not noisy. Approximation on all curve segments is reached in only one iteration and the average error is very small. The final reconstructed curve is shown in Figure 6.5. The reconstructed contour is smooth and approximates almost all of the sample points.

6.3. Feature Reconstruction

Feature reconstruction is implemented and tested under the Microsoft Visual C++ programming environment using ACIS R18 solid modeling libraries by Spatial. The GUI of the application has been implemented using HOOPS 16.20.

- Example

The feature region corresponding to the top of the screwdriver is fairly flat and spread out. Reconstruction of this feature is best carried out with covering, since it is simple to fit a surface to this feature. We first reconstruct the boundary curve of the feature region. Using this curve, we also slice the top of point cloud with a plane parallel to the boundary of the feature to obtain some points on the tip of the region.

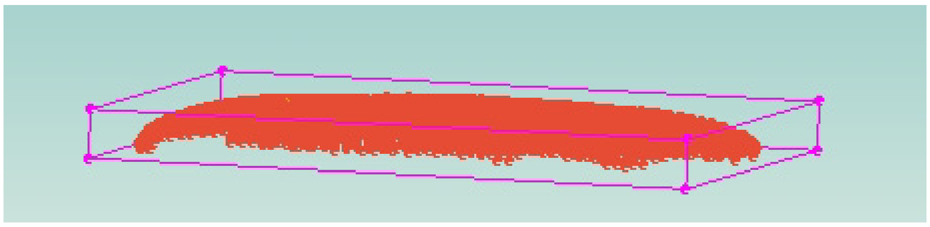


Figure 6.6 The top surface of the screwdriver

We perform covering using the points at the top as guides for the fitted surface. The resulting feature is shown is Figure 6.4.

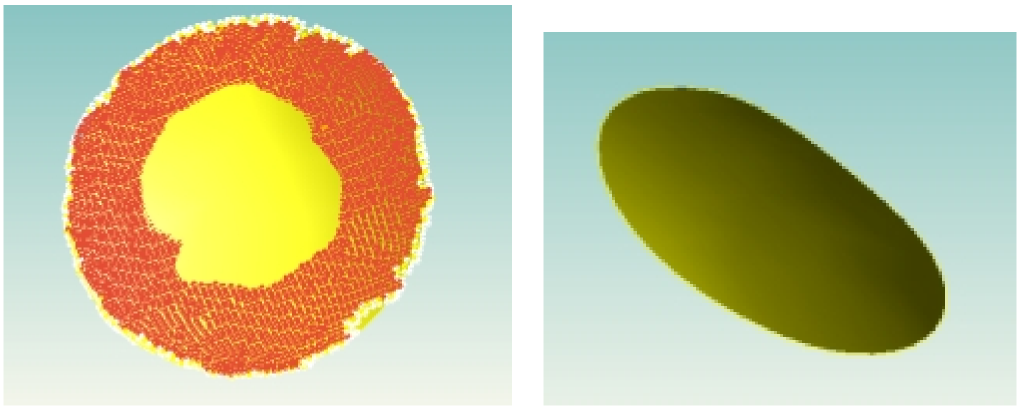


Figure 6.7 (left) The feature point cloud and its fitted surface and (right) the surface obtained by covering

CHAPTER 7. EXTENSIONS OF OUR APPROACH

In this work we have presented a reverse engineering scheme that detects and extracts features from a point cloud, reconstructs the boundaries of each feature region by fitting the borders with smooth piecewise rational curves and reconstructs the features using solid modeling functions to obtain parameterized, editable models. In this chapter we examine how this framework can be extended or adapted so that it can be applied to more advanced or specialized applications.

7.1. Custom design and redesign functionality

As described in a previous chapter, by using features and defining their parameters we obtain editable CAD models. We can achieve even higher levels of editability and flexibility if we combine the basic parameter and constraint definition with more local and global constraints. Local or global constraints are imposed on a model to enforce complex geometric structures and advanced functionality. Such constraints may be part of a feature or span a number of different features. By applying a system of constraints on a model and its features we can support custom design on a higher level, providing the capability to extract individual features from a model or a set of features and use them in the design process of a different model or for the redesigning of the current model.

7.1.1. Constraint definition

The constraints that we define on our features refer to distance, incidence, angle, parallelism, and perpendicularity. The primitives we define these constraints on are points, lines and planes. We use p_i to represent a point with coordinates (x_i, y_i, z_i) and l_i to represent a line. We use Pl_i to refer to a plane defined by its normal vector n_i and a point on the plane. A plane can also be defined by the equation:

$$ax + by + cz + d = 0$$

- Angle constraints

The angle a formed between two lines is calculated based on the dot product of the unit vectors t_i and t_j of the lines:

$$\cos a = t_i \cdot t_j \Rightarrow \cos a = t_{x_i} t_{x_j} + t_{y_i} t_{y_j} + t_{z_i} t_{z_j}$$

If the angle formed is 0 degrees, then the two lines are parallel, whereas if the angle is 90 degrees, then they are perpendicular.

In the case of the angle between two planes, we use the dot product of the normal vectors of the corresponding planes.

$$\cos a = n_i \cdot n_j \Rightarrow \cos a = n_{x_i} n_{x_j} + n_{y_i} n_{y_j} + n_{z_i} n_{z_j}$$

If the dot product is 1 then the planes are parallel, whereas if it is 0, then the planes are perpendicular. Any other value provides the angle formed by the two planes. To determine the relationship between two features in the CAD model, we use the normal vectors of the planes that pass through the centers of the features' corresponding bounding boxes and contain the two principal axes.

To determine the angle formed by a line and a plane, we use the unit vector of the line and the normal vector of the plane

$$\cos a = t_i \cdot n_j \Rightarrow \cos a = t_{x_i} n_{x_j} + t_{y_i} n_{y_j} + t_{z_i} n_{z_j}$$

- Distance constraints

The distance between two points $p_i(x_i, y_i, z_i)$ and $p_j(x_i, y_i, z_i)$ is determined by:

$$d^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2$$

The distance between two lines is useful when they are parallel. Their distance can be evaluated by picking a point p_j on one of the lines and calculating the distance from the other line so that the dot product of the distance vector and the line is 0.

$$d_{ij} \cdot li = 0$$

This approach can be used also for point line distance evaluation.

The distance between two planes can be computed when two planes are parallel. A point $p(x_j, y_j, z_j)$ on one of the planes (i.e. the origin of the plane) can be used to determine the distance from the other.

$$d(p, Pl_i) = \frac{ax_j + by_j + cz_j + d}{\sqrt{a^2 + b^2 + c^2}}$$

This formula can be used in general for point-plane distance computation.

The distance between a line and a plane can be determined by picking a point p_j on the line such that its distance from the plane Pl_i is d.

Incidence constraints are used to place primitives on primitives and are applied by setting the distance of the one primitive to the other as 0.

These constraints are applied on points, planes derived from the bounding boxes of features, as well as edges (lines) belonging to these planes or from paths. They are used to define the relationship between features of the cloud. For example, if two features are parallel then the planes passing through their corresponding bounding boxes will be parallel. Another example is two planes located at an angle, however edges belonging to the planes are parallel.

7.1.2. Geometric Constraint Solving

We assume that the system of constraints that were defined can be solved using an existing constraint problem solving method. A number of approaches have been suggested for solving constraint systems and we will briefly refer to the method characteristics [34].

Numerical Constraint Solvers

In numerical constraint solvers, the constraints are translated into a system of algebraic equations and are solved using iterative methods. To handle the exponential number of solutions and the large number of parameters, iterative methods require sharp initial guesses. Also, most iterative methods have difficulties handling overconstrained or underconstrained instances. The advantage of these methods is that they have the potential to solve large nonlinear system that may not be solvable using any of the other methods. All existing solvers more or less switch to iterative methods when the given configuration is not solvable by the native method. This fact emphasizes the need for further research in the area of numerical constraint solving.

Sketchpad [81] was the first system to use the method of relaxation as an alternative to propagation. Relaxation is a slow but quite general method. The Newton-Raphson method has been used in various systems [63, 70], and it proved to be faster than relaxation but it has the problem that it may not converge or it may converge to an unwanted solution after a chaotic behavior. For that reason, Juno [63] uses as initial state the sketch interactively drafted by the user. However, Newton-Raphson is so sensitive to the initial guess [8], that the sketch drafted must almost satisfy all constraints prior to constraint solving. A sophisticated use of the Newton-Raphson method was developed in [59], where an improved way for finding the inverse Jacobian matrix is presented. Furthermore, the idea of dividing the matrix of constraints into submatrices as presented in the same work has the potential of providing the user with useful information regarding the constraint structure of the sketch. Though this information is usually quantitative and nonspecific, it may help the user in basic modifications. To check whether a constraint problem is well-constrained, Chyz [21] proposes a preprocessing phase where the graph of constraints is analyzed to check whether a necessary condition is satisfied. The method is however quite expensive in time and it cannot detect all the cases of singularity. An alternative method to Newton-Raphson for geometric constraint solving is homotopy or continuation [4], that is argued in [51] to be more satisfactory in typical situations where Newton-Raphson fails. Homotopy, is global, exhaustive and thus slow when compared to the local and fast Newton's method [62], however it may be more appropriate for CAD/CAM systems when constructive methods fail, since it may return all solutions if designed carefully.

Constructive Constraint Solvers

This class of constraint solvers is based on the fact that most configurations in an engineering drawing are solvable by ruler, compass and protractor or using other less classical repertoires of construction steps. In these methods the constraints are satisfied in a constructive fashion, which makes the constraint solving process natural for the user and suitable for interactive debugging. There are two main approaches in this direction: Rule-constructive solvers and graph-constructive solvers.

Rule-constructive solvers use rewrite rules for the discovery and execution of the construction steps. In this approach, complex constraints can be easily handled, and extensions to the scope of the method are straightforward to incorporate [17]. Although it is a good approach for prototyping and experimentation, the extensive computations involved in the exhaustive searching and matching make it inappropriate for real world applications.

A method that guarantees termination, ruler and compass completeness and uniqueness using the Knuth-Bendix critical pair algorithm is presented in ([15], [74]). This method can be proved to confirm theorems that are provable under a given system of axioms [16]. A system based on this method was implemented in Prolog. Aldefeld in [3] uses a forward chaining inference mechanism, where the notion of direction of lines is imposed by introducing additional rules, and thus restricting the solution space. A similar method is presented in [82], where handling of overconstrained and underconstrained problems is given special consideration. Sunde in [80] uses a rule-constructive method but adopts different rules for representing directed and nondirected distances, giving flexibility for dealing with the solution selection problem. In [96], the problem of nonunique solutions is handled by imposing a topological order on three geometric objects. An elaborate description of a complete set of rules for 2D geometric constraint solving can be found in [90]. In their work, the scope of the particular set of rules is characterized. [48] presents an extension of the set of rules of [90], and provides a correctness proof based on the techniques of [35].

The graph-constructive approach has two phases. During the first phase the graph of constraints is analyzed and a sequence of construction steps is derived. During the second phase these construction steps are followed to place the geometric elements. These approaches are fast and more methodical. In addition, conclusions characterizing the scope of the method can be easily derived. A major drawback is that as the repertoire of constraints increases the graph-analysis algorithm needs to be modified.

Fitzgerald [32] follows the method of dimensioned trees introduced by Requicha [69]. This method allows only horizontal and vertical distances and it is useful for simple engineering drawings. Todd in [86] first generalized the dimension trees of Requicha. Owen in [64] presents an extension of this principle that includes circularly dimensioned sketches. DCM [24] is a system that uses some extension of Owen's method. [36] presents an elaborative graph-constructive method, with fast analysis and construction algorithms, and extensions for handling classes of nonsolvable, underconstrained and consistently overconstrained configuration

Propagation Methods

Propagation methods follow the approach met in traditional constraint solving systems. In this approach, the constraints are first translated into a system of equations involving variables and constants. The equations are then represented by an undirected graph which has as nodes the equations, the variables and the constants, and whose edges represent whether a variable or a constant appears in an equation. Subsequently, we try to direct the graph so as to satisfy all the equations starting from the constants. To accomplish this, various propagation techniques have been used but none of them guarantees to derive a solution and at the same time have a reasonable worst case running time. A review of these methods is provided in [74]. In a sense, the constructive constraint solvers can be thought of as a sub case of the propagation method (fixed geometric elements for constants and variable geometric elements for variables). However, constructive constraint solvers utilize domain specific information to derive more powerful and efficient algorithms.

Symbolic Constraint Solvers

In symbolic solvers, the constraints are transformed to a system of algebraic equations which is solved using methods from algebraic manipulation, such as Grobner basis calculation [18] or Wu's method[95]. Although, these methods are interesting from a theoretical viewpoint, their practical significance is limited, since their time and space complexity is typically exponential or even hyperexponential.

Hierarchical and hybrid approaches

A major result in analysis of constraint graphs by [42] in which an efficient method for detecting dense constraint subgraphs is described has enabled the solution of large systems of geometric constraints in 2, 3 or more dimensions. By using this result we can build efficient algorithm for solving arbitrary systems of geometric constraints. We first find a set of minimal disjoint dense constraint subgraphs. Each subgraph is then reduced in a supernode of high dimension and the method is applied recursively to the resulting graph. In this way we build a hierarchy of constraint graphs that is treated bottom up or top down based on the application. Interfeature 3D constraints result in systems of 3D constraints. Such systems are very hard to solve with graph constructive methods since there is not even a necessary and sufficient condition for well-constrainedness in 3D. By using the decomposition suggested by this approach we may breakdown the large geometric constraint system in a multitude of small systems with few variables. Such systems are usually easy to solve using global optimization with topological constraints to narrow down the root selection process.

In our system we have used a variation of the algorithm described in [41] in conjunction with various optimization techniques for solving efficiently the dense subsystems detected. We have also employed the ACIS constraint management tools.

7.2. Special applications

The framework suggested in this work can be extended and applied to applications with a more specific focus. An example of such an application is the re-engineering of jewellery, and more specifically, the reconstruction of traditional pierced Byzantine jewellery [75, 76].

7.2.1. Pierced Byzantine Jewellery

Pierced Byzantine jewellery are jewellery of a particular craftsmanship. The technique used to create them is a very sophisticated form of craftsmanship, as are the designs featured on them, and this technique has faded over the years. They are gold jewels with pierced designs that were made along the coastlines of the eastern Mediterranean Sea during the period 3rd –7th century A.D. Their originality is due to the particular processing technique that is used for their creation resulting in a special aesthetic effect. Pierced jewellery was created from thin sheets of gold. The designs were engraved on these sheets of gold with a thin sharp tool. After the outlining of the designs, holes following their shape were created and these were decorated with triangular carvings, using an iron chisel.

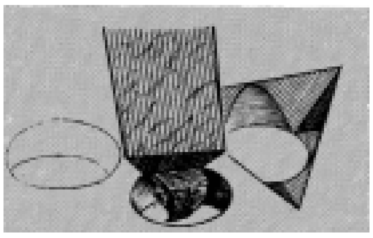


Figure 7.1 Using a chisel to create carvings around a hole.

The piercing technique was applied to various types of jewellery, such as necklaces, earrings, rings and bracelets. This technique was also used to create pierced crowns, pins used for holding tunics, belts and brooches. Jewellery such as earrings and pendants on the necklaces were usually shaped as rectangles, circles, hexagons, half moons (lunes) and teardrops. Many pieces of jewellery were decorated with sparsely-placed solid beads, or with a beadlike sequence wrapped around the piece of jewellery.

The designs used in Byzantine jewellery were specific. Usually, they were representations of nature or of human life. Frequently used nature inspired themes were different forms of birds, peacocks in particular, animals, especially dolphins, and plants, such as clovers and vine leaves. Many pieces of jewellery represented scenes combining nature and human. Byzantine jewellery had often representations of letters or religious symbols such as the cross. The letters used belonged either to the Greek or Latin alphabet and gave a personal touch to the jewellery.

Pierced Byzantine jewellery is interesting for many reasons. There is a limited number of such jewellery preserved until today. The technique used to create them is a very sophisticated form of craftsmanship, as are the designs featured on them, and this technique has faded over the years. Furthermore, their esthetic effect is unique. There are other techniques that result in jewellery with engraved designs, but they differ in the way that they are processed and the final aesthetic result. An example of such a technique is braiding plain or decorative wire to create the lace-like effect of pierced jewellery. The difference of the processing technique can be observed in the details of the designs and the back side of the jewellery. When the jewellery is pierced, the traces of the tool used for the piercing are apparent, and the back side of the jewellery is an ensemble of holes in a solid surrounding

7.2.2. An approach to designing pierced jewellery using feature elements

In [75, 76] ByzantineCAD, a feature-based CAD system suitable for the design of pierced Byzantine jewellery is presented. The system is automated and parametric meaning that the user-designer sets some parameter values and ByzantineCAD creates the jewellery model that corresponds to the specified values. This provides the designer with the ability to rapidly create custom-designed jewellery, based on the preferences of the customers, such as including their initials on a ring. ByzantineCAD introduces a feature-based and voxel-based approach to designing jewellery, through the definition of elementary structural elements (features) with specific attributes and properties that are used as building blocks to construct complex pierced designs.

ByzantineCAD uses voxel-based feature elements as building blocks for creating pierced jewellery. Each feature has characteristics that make it differ from other features, for example, a through hole, a pocket, a component forming an angle etc. By changing the parameters of the feature we can modify it to obtain an appropriate piece according to the jewellery that is being reconstructed. Each feature has a set of constraints that refer to its morphology, dimensions and behavior, in reference to itself and to other features. The feature elements are configured and combined using Boolean operations so as to create the CAD model (Figure 7.2).

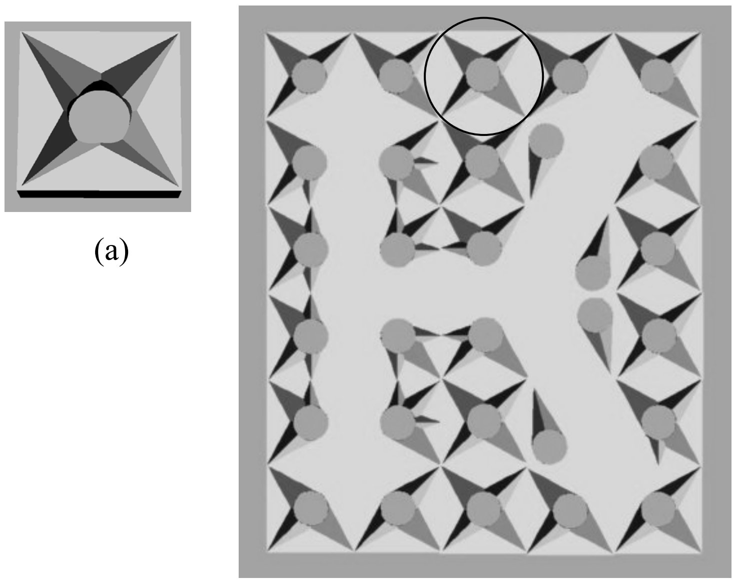


Figure 7.2 (left) A pierced voxel, (right) a pierced plate displaying the letter k created by combining pierced voxels

In ByzantineCAD a feature library of carved, pierced feature elements is defined in accordance to the craftsmanship used in traditional Byzantine jewellery. The design of pierced jewellery is made up of cylindrical holes that have carvings around them. Each hole with the corresponding carvings around it is considered for the purposes of reconstruction as a structural element (feature). Each feature is a solid made of a rectangular parallelepiped with a cylindrical hole and the corresponding carvings around the hole (Figure 7.2). According to the aesthetic rules that characterize traditional pierced jewellery, all feature elements have the same size but differ in the position of the hole and the carvings around it. The hole can be located either in the center of the parallelepiped or in the center of any of the four quarters. Note that, in terms of computer aided design and manufacturing, the cylindrical hole can be positioned anywhere in the rectangular parallelepiped; the above restriction follows from careful interpretation of the traditional artistic patterns used. Attributes of these feature elements are characteristics such as the number of carvings around the cylindrical hole, the position of the hole in the parallelepiped, the directions of the carvings and more. A large number of different features can be created by a hole and various carvings and, since not all of these feasible feature elements are valid for use in creating pierced designs, restrictions concerning the carving directions are defined based on aesthetic and artistic rules. A set of validity rules for features is defined, determining the number of carvings and their directions depending on the position of the hole in the voxel. For example, if the through hole is positioned in the center of the rectangular parallelepiped then, each carving starts from the hole and a carving cannot be directed towards the hole.

In the library of designs that has been developed for ByzantineCAD a subset of the set of features that satisfy the validity rules is used. Some characteristics of the feature elements that belong to the subset used in ByzantineCAD are a) the feature elements with holes located in the center have at most 5 carvings, and b) the feature elements with the hole in one of the quadrants do not have more than 3 carvings.

Each pierced design is a combination of features. Therefore, every design can be described using a “layout description file”, a file where the information needed to construct a specific pierced design is stored. Each design can be thought of as a two-dimensional matrix (Figure 7.2) whose entries correspond to feature elements. The layout description file determines the feature element that must be placed in each position of the matrix.

The layout description files of the letters of the Greek and Latin alphabet have been embedded in ByzantineCAD, along with some characteristic designs found in traditional Byzantine jewellery. It is possible for additional designs to be used by the system, as long as their layout description files are provided. The end-user can use an ordinary text editor to construct such designs by manually recording the sequence of features that form the design. In addition, the system provides a user-interface where the end-user can manually select valid feature elements and combine them to create new designs. After a new design is created, the user needs to store the sequence as a simple text file (the layout description file) and then this file can be imported to the ByzantineCAD library to enhance the repertoire of available pierced designs.

A pierced design is created by reading its corresponding layout description file. Each time the name of a feature is read, it is created, transformed (if necessary) and then translated to the proper location. The horizontal and vertical translations of the element are calculated using the equations:

$$x = h \times l$$

$$y = v \times k$$

where *x, y* are the horizontal and vertical translations respectively, *h* is the number of structural elements already placed horizontally in the current row, *v* is the number of structural elements already placed vertically in the current column and *l, k* are the height and length of the feature.

The pierced design on a piece of jewellery can be a sequence of individual designs. For instance, the design may be a sequence of letters forming a word. In this case, the process of creating the plate representing the word is the same as for a single design. The layout description files of each individual design are read in parallel and the plate is created row-wise (Figure 7.3). First the first line of the first letter is created, then the first line of the second letter is created and unioned with the first letter's first line and so on.

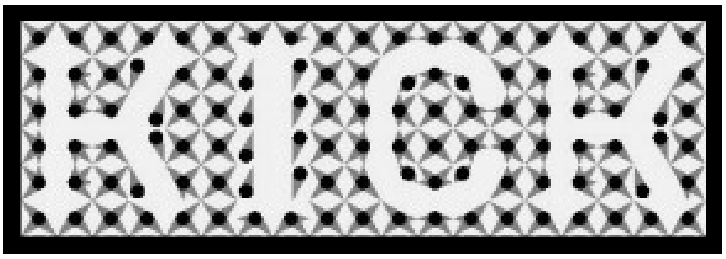


Figure 7.3 A complex solid plaque representing designs, i.e. letters or words, is sized and modified appropriately to construct custom-designed jewellery (i.e. ring).

Each time a feature element is placed, it is unioned with the previous ones. Eventually, a pierced plate representing the design is created.
ByzantineCAD is capable of designing rings, bracelets, necklaces and earrings. The end-user of ByzantineCAD defines the parameter values for the type of jewellery he would like to create. These parameter refer to type (ring, necklace), size, jewellery design, decoration (beaded border) etc. Earrings and necklaces are created in a number of different shapes, and are decorated with a beaded border (Figure 7.4(a)). Also, since in traditional Byzantine jewellery there were pieces in which solid non-pierced designs were placed in a pierced environment, ByzantineCAD has the capability of embedding solid designs in pierced surroundings (Figure 7.4(b)).

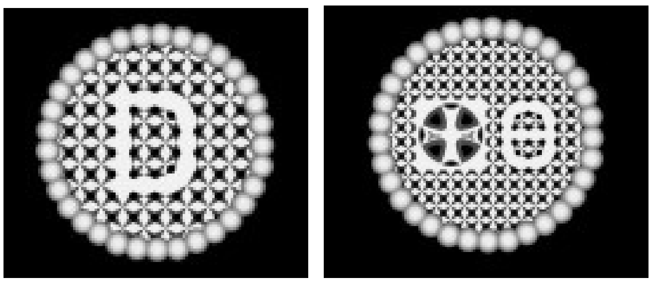


Figure 7.4 (left) An earring featuring the letter D, (right) a necklace featuring a solid design and a pierced design

An algorithm for scaling pierced patterns and designs is provided to enlarge pierced figures without altering the size of the feature elements used to construct them. Having this capability we may include, for instance, different font sizes in the same design. A pierced design is thought of as a 2-dimensional matrix whose every entry contains a feature element. Respectively, the scaled version of a design is a larger 2-dimensional matrix of features. The idea behind the scaling method is to gradually scan the design row by row using a sliding 2x2 window of feature elements, scale individually the 2x2 windows of the design and then integrate smoothly the scaled overlapping parts to create the scaled version of the design. The combinations of the features form different designs that can be categorized accordingly.

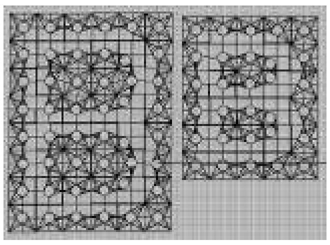


Figure 7.5 The letter B in its (left) scaled and (right) original form

The scaling that can be achieved is discrete, because of the need to preserve symmetries that may exist in the original design. For instance, letter B (Figure 7.5) is symmetric by a horizontal axis that goes through the middle of the design. A design is scaled by means of new rows and columns added to it. If we add only one new row to letter B, the letter becomes asymmetric, because if the row is added to the upper half or the lower half of the design, then the letter's shape is altered unintuitively. Also if it is added in the middle, the design becomes unproportionally thicker at the middle and therefore its original shape is modified. These restrictions are best expressed by the following rules for scaling upwards:

- avoid adding one row, or one column, and
- the number of rows and the number of columns must be integer

As a consequence of the first rule we choose to perform discrete scaling at a fixed factor. We choose a scale factor of 1.33 because it always results in adding two or more rows or columns. Therefore, from now on, we will refer to levels of scaling and not to a scaling factor. Level 1 corresponds to scaling the design by a factor of 1.33, Level 2 corresponds to a scale factor of 1.66 and so on.

A pierced design is represented by a 2-dimensional matrix whose entries correspond to feature elements. For instance let us consider the Level 1 scaling of a letter of font size $6 \times c$. When scaled to Level 1 a design is transformed from a $6 \times c$ matrix to an $8 \times k$ matrix (8 is the closest integer to $6 * 1.33= 7.98$) , where c and k are the number of columns of each matrix. The number of columns in the scaled design depends on the original number and is calculated in the same manner.

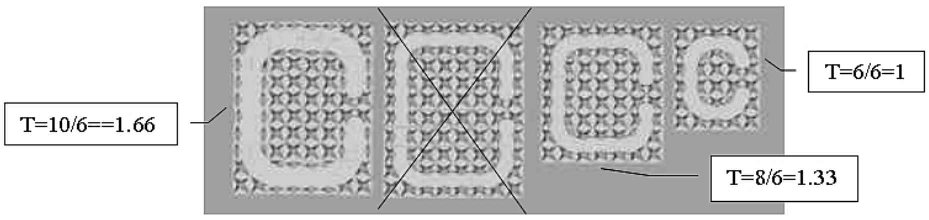


Figure 7.6 Different scaled versions of the letter C

We observe that while scaling a design, as the design gets larger, there is a need for thickening the engraved shape, so as to preserve its original form. In Figure 7.6 we see an original design of the C (rightmost design), which is scaled to two different levels (Level 1: 8 rows and Level 2: 10 rows). There are two different versions of Level 2 scaling. The crossed out version of letter C is not valid, because there is a distinct difference in the font style, compared to the original design. Therefore, the interior shape in Level 2 has to be thickened. We define as thickness factor T the ratio:

$$T = \frac{\text{Number of rows in scaled design}}{\text{Number of rows in original design}}$$

In the initial pierced design, the solid area that forms the shape that is created by two feature elements corresponds to approximately 90% of the whole area covered by the feature elements. Therefore the thickness factor is used in combination with this initial thickness to determine the thickness that the scaled design must have. The thickness H of the scaled design is determined by the product of the initial thickness H_0= 0.9 and the thickness factor. The quantum of the thickness increase is 0.5. Thus the discrete thickness H_d is expressed by the following equations:

$$H = H_0 T$$

$$H_d = \text{round}(2H)/2$$

For example, if the initial design consists of 6 rows and the second level scaled design consists of 10 rows, then the product of the thickness factor (10/6) and the initial thickness (0.9) is 1.5 which is the thickness the scaled design must have. If H_d= 1, the thickness of the curves inside the design is not altered. If H_d= 1.5 the thickness is increased by 50%, if H_d= 2 the shape thickness is doubled (100% increase) and so on.

The original design is scanned using a 2×2 window that starts scanning the design row-wise from the upper left corner. The design is scanned from left to right, and from top to bottom. At each step the window is shifted to the right by one position, and when an entire row has been scanned, the window is initialized at the beginning of the next row.

Before scanning and scaling, datum positions are marked in the scaled design matrix. We consider the structural elements positioned at North, South, East, West, South-East, South-West, North-West and North-East as our datum "points". These reference points are used for ensuring that symmetries are preserved and that the various proportions of the shapes within the design are maintained. Figure 7.7 depicts the eight structural elements used as reference points. When the number of rows of the scaled design is even reference points E and W are duplicated. Respectively, when the number of columns of the scaled design is even reference points N and S are duplicated.

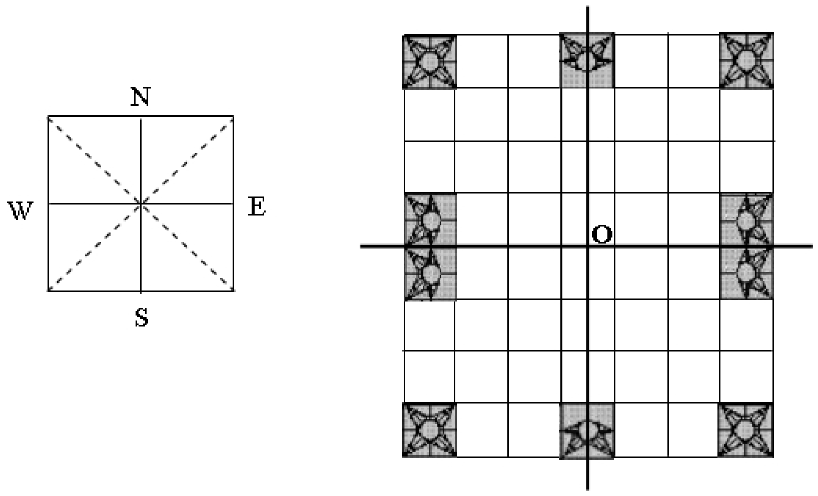


Figure 7.7 Datum positions are marked in the scaled design matrix and O is the center of the coordinate system created by the datum axes

The scaling algorithm can be described with the following steps:

Step1: Every time a window scan is performed, a combination of 4 feature elements is returned.

Step 2: This combination is scaled individually and placed appropriately in the scaled design matrix. The scaling of the 2×2 block of features is determined by the following principles:

- The relative position of the block in the original design should be maintained in the scaled design.
- The datum points should be respected.
- If the block contains part of a curve of a shape the corresponding curve should be scaled appropriately.

The size of the scaled combination is normally 3x3 for Level 1 scaling. However, according to the above principles the size of the scaled window may be reduced to 3x2 or 2x3 (one column or one row truncated), or 2x2 (one row and one column truncated). For the other levels of scaling, the size of the scaled combination is determined proportionally (Figure 7.8).

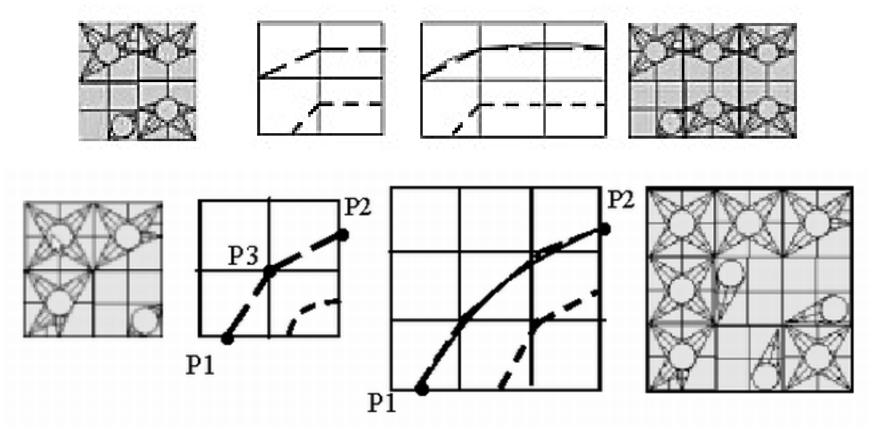


Figure 7.8 An example of (top) scaling a curve ending in a horizontal line and (bottom) scaling a curve

Step 3: The appropriate scaling for the specific combination determined in Step 2 is used to fill in the corresponding entries in the scaled matrix. This is placed in the new scaled matrix so as to overlap previous scaled windows. The overlapping is used to ensure that the connection among neighboring cells is a valid one. When two overlapping features are not the same, then we have a conflict which has to be resolved.

Overlapping and Conflict Resolution**:** Suppose S is the 2x2 sliding window which we obtain from step 1. After scaling this combination of features we obtain a scaled version S' of the sliding window. The scaled sliding window S' is placed in the scaled design matrix positioned in such a way so as to overlap previously scaled sliding windows. Suppose P' is the previously scaled window on the left of S', P'' is the previously scaled window located directly above S'. If S' is being placed in the first row of the design matrix then we overlap the last column of P' with the first column of S'. If S' is placed in one of the other rows then we not only overlap with P' but also with P'' by placing the first row of S' over the last row of P''.

After appropriately placing S' we check to see if any conflicts have occurred. A conflict occurs when overlapping elements do not match exactly. In this case, one of the overlapping elements must be picked to occupy the scaled design matrix position. A feature can always be found to occupy the conflicting position so that it conforms to the neighboring elements and continues the shape design correctly. A large number of cases are covered by the mirrored and rotated versions of a feature. The other cases are covered by using features that belong to the more general group of valid features, from which we only use a subset in ByzantineCAD. In cases where a feature with a hole in the center is used, we can ensure the continuation of a "curve" shape because we have all possible combinations of carvings. Otherwise, by using elements whose hole is not in the center, we can handle shapes where possibly there is a change in the shape angle-wise.

Step 4: The above steps are carried out row-wise until all of the design is scanned and scaled. If there are empty spaces in the scaled design matrix, then these are filled with the neutral feature.

The scaling algorithm can be summarized as follows:

```
for $i$ = 1 to n
    for $j$ = 1 to m
        step 1:    Consider the 2x2 window of feature elements: W[$i$, j]= [D[$i$, $j$], D[$i$+1, $j$], D[$i$, $j$+1], D[$i$+1, $j$+1]]
        step 2:    Determine the new magnified window $W_s$. This window will be 2x2, 2x3, 3x2 or 3x3 according to the category and position of the original window.
        step 3:    Update the corresponding positions of the new scaled matrix $D_s$ by placing the magnified window $W_s$ so as its upper left corner goes to [$i$, $j$]. If any such value conflicts with previous values of $D_s$ then integrate them so that the two overlapping windows join smoothly with each other
    end for
end for
step 4: Go through Ds searching for empty entries and fill them in with the neutral feature element.
```

where n is the number of rows and m is the number of columns, D is the matrix describing the original design, D_s is the matrix describing the scaled design, W is the sliding window, and W_s is the scaled sliding window.

7.2.3. Reverse engineering point clouds of traditional pierced jewellery

A 3D point cloud of traditional pierced jewellery is reconstructed by combining our feature detection approach with ByzantineCAD's voxel-based/feature-based modeling approach.

On one level, pierced feature voxels are scanned separately by a 3D laser scanner and their point clouds are reconstructed using our feature detection and extraction algorithm. The detected feature regions are used to define the properties of the feature. Each reconstructed feature can then be used as a feature element for parametric modeling, as described in ByzantineCAD.

In another re-engineering approach, on a point cloud corresponding to a piece of pierced jewellery, our feature detection method is used to detect regions corresponding to the particular characteristics of the object i.e carvings around a through hole. The feature regions obtained by feature detection are reconstructed using covering techniques to retrieve the surfaces corresponding to carving cuts. The reconstructed pieces are used in ByzantineCAD either for the definition of new feature elements or if they conform to the library already defined in ByzantineCAD, they are mapped to the specific voxels. The mapping is performed by examining the characteristics of the extracted regions. For example, if three triangular feature regions are neighbors and they are connected to a through hole, then their orientation in reference to the hole and the depth of the angles formed can help map the feature to the predefined library. From there on, the jewellery piece is reconstructed using the pierced voxel modeling scheme proposed in ByzantineCAD.

For example, in the case of a point cloud corresponding to the letter K designed as shown in Figure 7.2, the regions detected by the region growing method are grouped together to form the characteristics of a voxel (carvings) by grouping the regions with the through hole to which they are connected. By defining and applying constraints on the regions and by looking for symmetries we can help the mapping process by detecting repeated features in the point cloud.

Alternatively, pierced traditional jewellery can be re-engineered by reconstructing the patterns on the jewellery with embossing.

CHAPTER 8. CONCLUSIONS

A feature-based approach to reconstructing CAD models of objects from 3D point clouds has been presented. The proposed approach focuses on reconstructing objects of mechanical or freeform design as means of re-engineering, reproduction and redesign.

We present an automated method that detects features in a point cloud by exploiting point cloud morphology. Specifically, we use variations in the point concavity intensity along with variations in the mean local surface normal in a region growing method for segmenting the point cloud into subsets corresponding to features. This method detects features in point clouds of objects of both mechanical and freeform design. This region growing method provides, for each detected feature, a set of points corresponding to the feature region and a set of points corresponding to the boundary/boundaries of the feature. The capability of user-interaction, for post-processing of the detected regions, is provided. Regions can be merged and divided based on the preferences of the user. Also the user can modify the region growing parameters to reduce or increase the number of regions, ergo the number of features, detected.

We introduce a fast curve fitting algorithm which constructs rational curves with G^1 continuity, which adequately capture the morphology of the boundary of a 3D feature region. Convergence can be obtained in a couple iterations if we choose reasonable end point tangent vectors and the point cloud is not noisy, otherwise a few iterations are required to adjust the weights and to adapt the parameterization. This is an automated process in which the only user intervention supported is the modification of the angle used for dividing the boundary into curve segments.

We evaluate how solid modeling techniques can efficiently be applied to our feature regions for means of reconstruction, to obtain an editable CAD model. Parameters and constraints are defined on the features to serve this purpose.

In general, the work presented in this thesis provides a framework for feature based reverse engineering. This framework has used successfully in jewellery redesign. Parts of our suite of technique have been used in different applications. For example, our feature detection and extraction method has been used to detect features and feature borders in objects for use by morphing software. The detected features are used for alignment and matching. Our framework augmented with a powerful geometric constraint solving system provides advanced editing capabilities, and high-level custom design and redesign.

Reasonable restriction of the required user- interaction is an aspect that needs further research. Specifically, feature reconstruction is implemented in a semi-automated way, where parameter and constraint definition is carried out mainly by the user whereas the slicing and reconstruction is performed automatically. Methods for automatic detection of symmetries and constraints have been developed. Further research can be carried out on how these methods can be integrated and augmented to fit our suggested framework. Also the level of automation that is achievable without compromising the effectiveness of the reconstruction process should be further investigated.

Another interesting research aspect is shape based retrieval of features. Considering that the features detected by our re-engineering approach are described by their construction methods (boundary contours, slice contours, paths), these parameters can be used as shape descriptors for retrieval of features from 3D feature libraries.

REFERENCES

1. D. Agbodan, D. Marcheix, and G. Pierra. Persistent Naming for Parametric Models, in WSCG 2000.

2. Aim@Shape, Aim@Shape Shape Repository V4.0, Dept. Of Genova, Institute for Applied Mathematics and Information Technologies, http://shapes.aimatshape.net. AIM@SHAPE Project.

3. B. Aldefeld, Variation of Geometries Based on a Geometric-Reasoning Method, Computer-Aided Design, Vol. 20(3), pp. 117-126, 1988.

4. E.L. Allgower and K. Georg, Continuation and Path Following, Acta Numerica, pp. 1-64, 1993.

5. N. Amenta, M. Bern, and M. Kamvysselis, A New Voronoi-Based Surface Reconstruction Algorithm, Siggraph, 1998.

6. C.K. Au and M.M.F. Yuen, Feature-Based Reverse Engineering of Mannequin for Garment Design, Computer-Aided Design, Vol. 31, pp. 751-759, 1999.

7. C.B. Barber, D.P. Dobkin, and H.T. Huhdanpaa, The Quickhull Algorithm for Convex Hulls, ACM Transactions on Mathematical Software, Vol. 22(4), pp. 469-483, 1996.

8. P.L. Beaty, P.A. Fitzhorn, and G.J. Herron, Extensions in Variational Geometry That Generate and Modify Object Edges Composed of Rational Bezier Curves, Computer-Aided Design, Vol. 26(2), pp. 98-108, 1994.

9. P. Benko, G. Kos, T. Varady, L. Andor, and R.R. Martin, Constrained Fitting in Reverse Engineering, Computer-Aided Geometric Design, Vol.19, pp.173-205, 2002.

10. P. Benko, R.R. Martin, and T. Varady, Algorithms for Reverse Engineering Boundary Representation Models, Computer-Aided Design, Vol. 33(11), pp. 839-851, 2001.

11. F. Bernardini, C.L. Bajaj, J. Chen, and D.R. Schikore, Automatic Reconstruction of 3D CAD Models from Digital Scans, International Journal of Computational Geometry & Applications, Vol. 9, pp. 327-369, 1999.

12. Blender, Blender Suite, Open Source Suite, http://www.blender.org, Blender Foundation.

13. J. Bloomenthal. Medial-Based Vertex Deformation, in Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, San Antonio, Texas, USA, pp. 147-151, 2002.

14. C. Bradley and B. Currie, Advances in the Field of Reverse Engineering, Computer Aided Design and Applications, Vol. 2(5), pp. 697-706, 2005.

15. B. Bruderlin. Constructing Three-Dimensional Geometric Objects Defined by Constraints, In Workshop on Interactive 3D Graphics, ACM, 1986.

16. B. Bruderlin, Using Geometric Rewrite Rules for Solving Geometric Problems Symbolically, Theoretical Computer Science, Vol. 116, pp. 291-303, 1993.

17. B. Bruderlin and D. Roller, Geometric Constraint Solving and Applications. Springer Verlag, 1998

18. B. Buchberger, Grobner Bases: An Algorithmic Method in Polynomial Ideal Theory, in Multidimensional Systems Theory, N.K. Bose, Editor, D. Reidel Publishing Company, pp. 184-232, 1985

19. X. Chen and C.M. Hoffmann, On Editability of Feature-Based Design, Computer Aided Design, Vol. 27(12), pp. 905-914, 1995.

20. B. Choi, H. Shin, Y.I. Yoon, and J. Lee, Triangulation of Scattered Data in 3D Space, Computer Aided Design, Vol. 20(5), pp. 239-248, 1988.

21. W. Chyz, Constraint Management for CSG, MSc Thesis, MIT,1985

22. N.D. Cornea, M.F. Demirci, D. Silver, A. Shokoufandeh, S.J. Dickinson, and P.B. Kantor. 3d Object Retrieval Using Many-to-Many Matching of Curve Skeletons,

in IEEE International Conference on Shape Modeling and Applications, Boston USA, 2005.

23. Cyberware, Cyberware Rapid 3d Scanners - Desktop 3d Scanner Samples. 1999.

24. D-Cubed, 68 Castle Street, Cambridge, CB3 OAJ, England. The Dimensional Constraint Manager, Version 2.7, June 1994.

25. J. Daniels, L. Ha, T. Ochotta, and C. Silva. Robust Smooth Feature Extraction from Point Clouds, in IEEE International Conference on Shape Modeling and Applications (SMI '07). 2007.

26. H.J. De St. Germain, S.R. Stark, W.B. Thompson, and T.C. Henderson. Constraint Optimization and Feature-Based Model Construction for Reverse Engineering, In Proceedings of the ARPA Image Understanding Workshop, 1996.

27. K. Demarsin, D. Vanderstraeten, T. Volodine, and D. Roose, Detection of Closed Sharp Edges in Point Clouds Using Normal Estimation and Graph Theory, Computer Aided Design, Vol. 39, pp. 276-283, 2007.

28. T. Dey and S. Goswami. Tight Cocone: A Water-Tight Surface Reconstructor, in ACM Symposium on Solid and Physical Modeling, Seattle, Washington, USA., pp. 127-134, 2003.

29. D. Dimitrov, C. Knauer, K. Kriegel, and G. Rote. Upper and Lower Bounds on the Quality of the PCA Bounding Boxes, in Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision - WSCG '07, Plzen, Czech Republic, pp. 185-192, 2007.

30. L. Fang and D. Gossard, Multidimensional Curve Fitting to Unorganized Data Points by Nonlinear Minimization, Computer Aided Design, Vol. 27(1), pp. 48-58, 1995.

31. R.B. Fisher, Applying Knowledge to Reverse Engineering Problems, Computer-Aided Design, Vol. 36, pp. 501-510, 2004.

32. W.J. Fitzgerald, Using Axial Dimensions to Determine the Proportions of Line Drawings in Computer Graphics, Computer Aided Design, Vol. 13(6), pp. 377-382, 1981.

33. S. Flory and M. Hofer, Constrained Curve Fitting on Manifolds, Computer Aided Design, Vol. 40, pp 25-34, 2008.

34. I. Fudos, Constraint Solving for Computer Aided Design, PhD Thesis, Dept of Computer Sciences, Purdue University,1995

35. I. Fudos and C.M. Hoffmann, Correctness Proof of a Geometric Constraint Solver, International Journal of Computational Geometry & Applications, Vol. 6(4), pp. 405-420, 1996.

36. I. Fudos and C.M. Hoffmann, A Graph-Constructive Method to Solving Systems of Geometric Constraints, ACM Transactions on Graphics, Vol. 16(2), pp. 179-216, 1997.

37. Y. Gong, J. Chen, T. Jin, and S. Tong. Feature and Constraints-Based to Reconstruct Prototype Methods and Application, in Proceedings of the IEEE International Conference on Mechatronics & Automation, Niagara Falls, Canada, 2005.

38. S. Gottschalk, Collision Queries Using Oriented Bounding Boxes, PhD Thesis, Dept of Computer Science, University of North Carolina, 2000

39. S. Gumhold, X. Wang, and R. Macleod. Feature Extraction from Point Clouds, in Proceedings of the 10th International Meshing Round Table, pp.293-305, 2001.

40. C.M. Hoffmann and R. Joan-Arinyo, On User-Defined Features, Computer Aided Design, Vol. 30(5), pp 321-332, 1998.

41. C.M. Hoffmann and R. Juan, Erep - an Editable, High-Level Representation for Geometric Design and Analysis, in Geometric Modeling for Product Realization, Wilson P., Wozny M., Pratt M., eds, North Holland, 1993

42. C.M. Hoffmann, A. Lomonosov, and M. Sitharam, Finding Solvable Subsets of Constraint Graphs, Smolka. G., Editor, pp. 463-477, 1997.

43. H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle, Surface Reconstruction from Unorganized Points, Computer Graphics, Vol. 26(2), pp. 71-78, 1992.

44. J. Hoschek and D. Lasser, Fundamentals of Computer Aided Geometric Design, ed. A. Peters. 1993.

45. J. Huang and C. Menq, Automatic Data Segmentation for Geometric Feature Extraction from Unorganized 3-D Coordinate Points, IEEE Transactions on Robotics and Automation, Vol. 17(3), pp. 268-279, 2001.

46. K.A. Ingle, Reverse Engineering, McGraw-Hill, 1994.

47. IPOPT - Interior Point Optimizer, http://projects.coin-or.org/Ipopt

48. R. Juan-Arinyo and A. Soto, A Rule-Constructive Geometric Constraint Solver, Technical Report LSI-95-25-R, Universitat Politecnica de Catalunya, 1995.

49. J. Kantz, Application of Sweeping Techniques to Reverse Engineering, MSc Thesis, Department of Computer and Information Science, University of Michigan - Dearborn, 2003

50. J. Kripac. A Mechanism for Persistently Naming Topological Entities in History-Based Parametric Solid Models, in Solid Modeling '95. Salt Lake City Utah USA, 1995

51. H. Lamure and D. Michelucci. Solving Geometric Constraints by Homotopy, In Proc. Third Symposium on Solid Modeling and Applications, Salt Lake City, pp. 263-269, 1995.

52. F.C. Langbein, A.D. Marshall, and R.R. Martin, Choosing Consistent Constraints for Beautification of Reverse Engineered Geometric Models, Computer-Aided Design, Vol. 36, pp. 261-278, 2004.

53. F.C. Langbein, B.I. Mills, A.D. Marshall, and R.R. Martin. Finding Approximate Shape Regularities in Reverse Engineered Solid Models Bounded by Simple Surfaces, In Proceedings of the 6th Symp. Solid Modeling & Applications, ACM, pp. 206-215, 2001.

54. I.-K. Lee, Curve Reconstruction from Unorganized Points, Computer Aided Geometric Design, Vol. 17, pp. 161-177, 2000.

55. K. Lee, Principles of CAD/CAM/CAE Systems, ed. Addison-Wesley. 1999.

56. J.M. Lien and N.M. Amato, Approximate Convex Decomposition, TR03-001, PARASOL LAB, Department of Computer Science, Texas A&M University, 2003

57. J.M. Lien and N.M. Amato, Approximate Convex Decomposition of Polyhedra, TR05-001, PARASOL LAB, Department of Computer Science, Texas A&M University, 2005.

58. J.M. Lien and N.M. Amato. Simultaneous Shape Decomposition and Skeletonization, In Proceedings of the ACM Solid and Physical Modeling Symposium (SPM), pp. 219-228, 2006.

59. R. Light and D. Gossard, Modification of Geometric Models through Variational Geometry, Computer Aided Design, Vol. 14(4), pp. 209-214, 1982.

60. D. Marcheix and G. Pierra. A Survey of the Persistent Namine Problem, in SM '02, Saarbrucken,Germany, June 17-21, 2002.

61. B.I. Mills, F.C. Langbein, A.D. Marshall, and R. Martin. Approximate Symmetry Detection for Reverse Engineering, in ACM Symposium on Solid and Physical Modeling, Ann Arbor, Michigan, United States, pp. 241-248, 2001.

62. A. Morgan, Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems, Prentice Hall Inc., 1987

63. G. Nelson. Juno, a Constraint-Based Graphics System, In SIGGRAPH, San Francisco, USA, pp. 235-243, 1985.

64. J.C. Owen. Algebraic Solution for Geometry from Dimensional Constraints, In ACM Symp. Found. of Solid Modeling, Austin, TX, 1991.

65. L. Piegl and W. Tiller, The NURBS Book, ed. D. Rogers., Springer-Verlag, 1997

66. H. Pottmann, S. Leopoldselder, and M. Hofer. Approximation with Active Bspline Curves and Surfaces, in Proc of the Pacific Graphics IEEE, pp. 8-25, 2002.

67. S. Raghothama and V. Shapiro, Boundary Representation Deformation in Parametric Solid Modeling, ACM Transactions on Graphics, Vol. 17, pp. 259-286, 1998.

68. G. Renner, A Method of Shape Description for Mechanical Engineering Practice, Computers in Industry, Vol. 3, pp. 137-142, 1982.

69. A. Requicha, Dimensionining and Tolerancing, in Technical report PADL TM-19, Production Automation Project, University of Rochester, 1977.

70. D. Serrano and D. Gossard. Combining Mathematical Models and Geometric Models in Cae Systems, In Proc. ASME Computers in Eng. Conf., Chicago, pp. 277-284, 1986.

71. J.J. Shah and M. Mantyla, Parametric and Feature-Based CAD/CAM. John Wiley & Sons Inc, 1995.

72. D. Sheehy, C. Armstrong, and D. Robinson, Shape Description by Medial Axis Construction, IEEE Transactions on Visualization and Computer Graphics, Vol. 2(1), pp. 62-72, 1996.

73. SINTEF, SISL - Sintef Spline Library, http://www.sintef.no/content/page1____5470.aspx

74. W. Sohrt, Interaction with Constraints in Three-Dimensional Modeling, MSc Thesis, Dept of Computer Science, The University of Utah, 1991

75. V. Stamati and I. Fudos, A Parametric Feature-Based Cad System for Reproducing Traditional Pierced Jewellery, Computer Aided Design, Vol. 37(4), pp. 431-449, 2005.

76. V. Stamati, I. Fudos, S. Theodoridou, C. Edipidi, and D. Avramidis. Using Poxels for Reproducing Traditional Pierced Byzantine Jewellery. in Proceedings of the Computer Graphics International, Crete, Greece, 2004.

77. Stanford, The Stanford Bunny - the Stanford 3d Scanning Repository. 1993.

78. D.W. Storti, G.M. Turkiyyah, M.A. Ganter, C.T. Lim, and D.M. Stal. Skeleton-Based Modeling Operations on Solids, Solid Modeling '97, Atlanta GA USA., 1997.

79. X. Sun, P. Rosin, R. Martin, and F.C. Langbein, Fast and Effective Feature-Preserving Mesh Denoising, IEEE Transactions on Visualization and Computer Graphics, Vol. 13(5), pp. 925-938, 2007.

80. G. Sunde, Specification of Shape by Dimensions' and Other Geometric Constraints, in Geometric Modeling for CAD Applications. M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, eds, North Holland, pp. 199-213, 1988

81. I. Sutherland. Sketchpad, a Man-Machine Graphical Communication System. In Proc. of the Spring Joint Compo Conference, 1963.

82. H. Suzuki, H. Ando, and F. Kimura, Variation of Geometries Based on a Geometric-Reasoning Method, Computers & Graphics, Vol. 14(2), pp. 211-224, 1990.

83. L. Szobonya and G. Renner. Construction of Curves and Surfaces Based on Point Clouds, In Proc. First Hungarian Conference on Computer Graphics and Geometry, Budapest Hungary, 2002.

84. W.B. Thompson, H. De St. Germain, T.C. Henderson, and J.C. Owen. Constructing High-Precision Geometric Models from Sensed Position Data, In Proceedings ARPA Image Understanding Workshop. 1996.

85. W.B. Thompson, J.C. Owen, H.J. De St. Germain, S.R. Stark, and T.C. Henderson, Feature-Based Reverse Engineering of Mechanical Parts, IEEE Transactions on Robotics and Automation, Vol.15(1), pp. 57-66, 1999.

86. P. Todd. A K-Tree Generalization That Characterizes Consistency of Dimensioned Engineering Drawings, SIAM J. DISC. MATH, Vol. 2(2), pp. 255-261, 1989.

87. W.-D. Ueng, J.-Y. Lai, and Y.-C. Tsai, Unconstrained and Constrained Curve Fitting for Reverse Engineering, International Journal of Advances Manufacturing Technology, 33, pp. 1189-1203, 2007.

88. T. Varady, M. Facello, and Z. Terek, Automatic Extraction of Surface Structures in Digital Shape Reconstruction, Computer Aided Design, Vol. 39, pp. 379-388, 2007.

89. T. Varady, R.R. Martin, and J. Cox, Reverse Engineering of Geometric Models - an Introduction, Computer-Aided Design, Vol. 29(4), pp. 255-268, 1997.

90. A. Verroust, F. Schonek, and D. Roller, Rule-Oriented Method for Parameterized Computer-Aided Design, Computer Aided Design, Vol. 24(10), pp. 531-540, 1992.

91. J. Vollmer, R. Mencl, and H. Muller, Improved Laplacian Smoothing of Noisy Surface Meshes, Computer Graphics Forum, Vol. 18(3), pp.131-138, 1999.

92. J. Wang, M.M. Oliveira, and A. Kaufman. Reconstructing Manifold and Non-Manifold Surfaces from Point Clouds, in Visualization 2005.

93. W. Wang, H. Pottmann, and Y. Liu, Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization, ACM Transactions on Graphics, Vol. 25(2), pp. 214-238, 2006.

94. A. Werner, K. Skalski, S. Piszczatowski, W. Swieszkowski, and Z. Lechniak, Reverse Engineering of Free-Form Surfaces, Journal of Materials Processing Technology, Vol. 16, pp. 128-132, 1998.

95. W.T. Wu, Basic Principles of Mechanical Theorem Proving in Elementary Geometries, Journal of Automated Reasoning, Vol. 2, pp. 221-252, 1986

96. Y. Yamaguchi and F. Kimura, A Constraint Modeling System for Variational Geometry, in Geometric Modeling for Product Engineering, M. J. Wozny, and K. Preiss, eds, Elsevier Science Publishers B.V. (North Holland), pp. 221-233, 1990

97. Z. Yang and Y. Chen, A Reverse Engineering Method Based on Haptic Volume Removing, Computer-Aided Design, 37, pp. 45-54, 2005

98. H.-T. Yau and J.-S. Chen, Reverse Engineering of Complex Geometry Using Rational B-Splines. International Journal of Advances Manufacturing Technology, Vol.13, pp. 548-555, 1997

99. S. Yoshizawa, A.G. Belyaev, and H.-P. Seidel. Free-Form Skeleton-Driven Mesh Deformation, in Proceedings of the eighth ACM Symposium on Solid Modeling and Applications, Seattle, Washington U.S.A., 2003

PUBLISHED WORK

- V. Stamati and I. Fudos, A Parametric Feature-based CAD System for Reproducing Traditional Jewellery, Computer Aided Design & Applications, 2004, 559-568 (also presented in CAD '04)
- V. Stamati, I. Fudos, S. Theodoridou, C. Edipidi, and D. Avramidis Using Poxels for reproducing traditional pierced Byzantine jewellery. In Computer Graphics International 2004, Crete, Greece, June 16-19
- V. Stamati and I. Fudos, A parametric feature-based CAD system for reproducing traditional pierced jewellery, Computer Aided Design 2005, 37(4), 431-449
- V. Stamati and I. Fudos, "A feature-based CAD approach to jewellery re-engineering", Computer-aided Design and Applications, Vol. 2, No.1-4, 1-10, 2005 (also presented in CAD '05)
- V. Stamati and I. Fudos, "A Feature-Based Approach to Re-engineering Objects of Freeform Design by Exploiting Point Cloud Morphology", in Proceedings of SPM 2007: ACM Symposium on Solid and Physical Modeling, Beijing China, pp. 347-353, June 2007
- I. Fudos and V. Stamati, "Constraint-based and Feature-based CAD Systems and Applications" for book: Computer-Aided Design Research and Development, NovaPublishers (accepted – in print), 2008
- V. Stamati and I. Fudos, "On Reconstructing 3D Feature Boundaries", Computer Aided Design and Applications, Vol. 5, No.1-4, 316-324, June 2008 (also presented in CAD '08)

Technical Reports

- V. Stamati and I. Fudos, CAD/CAM Methods for Reverse Engineering: A Case Study of Reengineering Jewellery, Technical Report 2004-15, Computer Science Dept, University of Ioannina 2004
- I. Fudos, V. Stamati and A. Protopsaltou, An Approach to Geometric Constraint Solving for CAD Representations, Technical Report TR 2004-17, Computer Science Dept., University of Ioannina, 2004

SHORT CV

Vicky Stamati was born on July 16, 1977 in Calgary, Alberta, Canada. She received her diploma in Computer Science from the Computer Science Dept. of the University of Ioannina, Greece in 2001 and her M.Sc. in Computer Science from the same University in 2003. Her research interests include parametric feature-based design, reverse engineering and solid modeling.