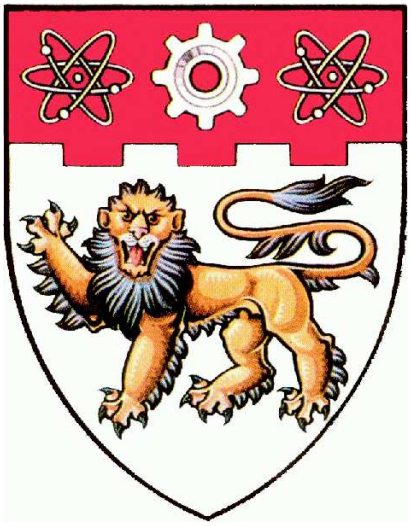
NANYANG TECHNOLOGICAL UNIVERSITY



SKETCH-BASED 3D MODELING AND RECONSTRUCTION

BY

WANG KAI

A THESIS SUBMITTED TO THE SCHOOL OF COMPUTER ENGINEERING

OF THE NANYANG TECHNOLOGICAL UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY (PHD)

2013

Abstract

This research investigates the use of sketching techniques in 3D modeling and reconstruction to enhance various processes such as 3D model creation, editing, analyzing and computing. 3D geometric models have been widely used in various applications from industrial design to digital entertainment. However, creating and processing 3D models with traditional 2D WIMP-style (Windows, Icon, Menu, Pointer) interfaces are generally tedious and time-consuming. On the other hand, sketching interfaces are emerging as a new approach to enhance user interactions for various design activities. The sketching interfaces are expected to provide flexible and intuitive interactions between computers and users. To effectively incorporated the sketching techniques into 3D modeling and reconstruction, some fundamental theories and supporting algorithms should be developed. The goal of this research is thus to investigate how to create intuitive and effective sketching interfaces and meanwhile to develop sketch-based techniques for creating and editing 3D geometric models.

Firstly, we develop a reference plane assisted sketching interface for freeform shape design, in which users can create and edit 3D models through simple sketches. To regularize and interpret the user input properly when sketching the shape of the 3D object, we introduce some rules for the strokes into the system, which consider both the semantic meaning of the sketched strokes and human psychology. To make 3D location and orientation of strokes meaningful and intuitive within 2D interface, we propose to perform the modeling operations with reference to some auxiliary planes, which are automatically constructed based on the user sketches or default setting. The use of reference planes helps to make the sketch-based modeling system more intuitive and easy to use.

Secondly, we propose a progressive modeling approach to allow users to create a 3D model with arbitrary topology through iterative sketches, by taking both the existing sketched curves and the up-to-date reconstructed model as references. In this way, users get aware of the shape of the sketched model, which helps to avoid unexpected modeling result. To effectively support this modeling approach, we present a CSG-based surface

reconstruction algorithm that enables gradual shape updates and produces models with a single connected component when the user iteratively adds new sketches. By using this progressive modeling approach in the sketching interface, the creation function will become more powerful in producing a desired 3D shape by sketching iteratively from scratch, and ambiguities on perception of the sketched shape can be avoided to a large extent.

Thirdly, we propose an edge-based flexible mesh deformation algorithm to effectively support the editing function in our sketch-based modeling system. An edge-based graph is proposed for a triangular mesh, which makes more points get involved in the evaluation of shape of the mesh surface and thus renders more accurate computation results. Based on the edge-based graph, we develop a flexible mesh deformation algorithm which gives users flexibility to adjust the deformation effects between local shape preservation and global smoothness through manipulating a simple balance parameter. This scheme is also applied to local parts of a mesh, to approximate the deformation effect of real-world objects with non-uniformly distributed materials. In this way, the deformation process in the sketch-based modeling will become more flexible and effective for users to produce a satisfactory modeling result.

Finally, we apply the sketching technique to tackle the problem of surface reconstruction from general cross section curves. A new framework is proposed to build a smooth mesh surface from cross sections with arbitrary orientations. The framework consists of two parts. The first part generalizes the algorithm in our progressive modeling to reconstruct surfaces interpolating the input cross sections with arbitrary orientations and having regular local shapes. The second part allows users to edit the local topology of the reconstructed model through simple sketches, if the initial one is not satisfactory. It makes use of the intermediate information obtained in the first part and thus avoids complicated analysis and computations. Our framework has the unique property of producing a manifold surface with only one connected component, which is usually desired in practice. It can be used in many geometric modeling tasks, including sketch-based modeling.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Zheng Jianmin, for his patient instructions and constant support on this research. His great insight and knowledge always guide me stepping forward and getting out of confusions and frustrations, and his serious attitude helps me to keep on improving the quality of the research work. I believe the things I learned from him will benefit me a lot in my future development. I also thank Prof. Seah Hock-Soon for his instructions and support on my research. I deeply appreciate their encouragement and sincerely hope that our cooperation will continue in the future.

I should appreciate the financial support for this research provided by the National Research Foundation grant, which is administrated by the Media Development Authority Interactive Digital Media Programme Office, and the space and equipment provided by GameLab, Nanyang Technological University.

It is a great pleasure to have the useful cooperations, discussions and funs with my following colleagues and friends in the past four years: Dr. Wang Yimin, Dr. Pan Jianjiang, Dr. Chen Xiaodiao, Dr. Chen Wenyu, Dr. Xin Shiqing, Dr. Li Xin, Dr. Cao Juan, Ma Yuewen, Zhou Hailing, Wu Xiaoqun, Li Yusha, Zhao Ming, Chen Shuangmin, Zhang Wenjing, and Zhang Yuzhe.

Deep thanks to Prof. Wu Zhongke and Prof. Tian Feng for providing me the chance to come to Singapore and see the world. I also appreciate the encouragement and valuable opinions on my career development from Prof. Zhao Zhiwen and Prof. Yao Li.

I would like to extend the deepest appreciations to my parents and my wife, for their constant and reliable support which gives me the faith and courage to face all the challenges in my life.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Geometric models are the kernels of many scientific and engineering subjects. They are widely used in various applications including product design, digital entertainment, biology, archeology and so on [27]. Correspondingly, there is a strong need for development of modeling and reconstruction techniques to create, edit, analyze and compute 3D models. Although there have been a lot of techniques and devices such as image-based modeling [44, 103] and 3D scanner techniques [95, 168] developed to build virtual 3D models, most 3D shapes still need to be constructed manually via various modeling tools by the user, either from scratch or by editing existing models.

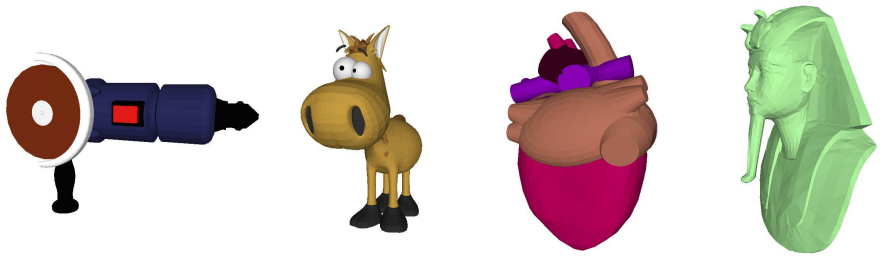


Figure 1.1: Geometric models used in various applications. These models are courtesy of the INRIA Gamma team research database [7].

Traditional tools for 3D modeling and reconstruction such as 3ds Max [2], Maya [3] and

so on are mainly designed for expert users to precisely construct sophisticated models [30]. They are difficult to use. One must spend a long time learning the tools and is expected to work on a task for a period of time to obtain a final result. This seems too complicated for novice users to master. These tools are also difficult for professional users in the initial design phase. Designers still need to work with pen and paper first and then move on to the computational tools when the design is fixed.

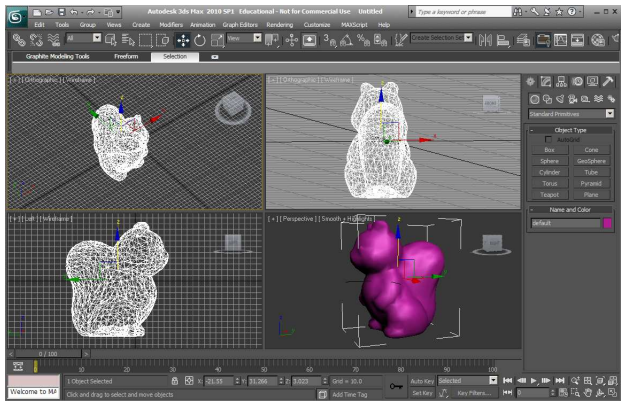


Figure 1.2: 3D modeling using the 3ds Max software [2].

Advances in information technology have enabled many systems to extend human's abilities in design and creative work. Modern cognition study indicates that human has two cognitive modes: experimental cognition and reflective cognition [118]. The former corresponds to skilled behavior and the latter accounts for intellectual work. Tano et al. [149] show that while most of current computer graphics (CG) and virtual reality (VR) tools are good for experimental cognition and poor for reflective cognition, sketch-based tools can promote reflective cognitive activities. On the other hand, with the rapid development of pen-based computer hardware, digital sketching interface emerges as a powerful way for users to interact with the computer and do various 3D modeling and reconstruction tasks.

Sketch-based modeling refers to the process of producing 3D models from user sketched curves or strokes. Instead of requiring the users to work with buttons, menus, icons and pointers, it allows the direct expression of their thoughts in the form of freehand sketching. The goal of the sketching interfaces is to solve the problem of accessibility by reducing

the amount of explicit control of the users to simplify the user interface dramatically. The sketch-based technique makes 3D modeling and reconstruction tasks accessible to novice users. Meanwhile, it can also be used by experts in their initial design stage. With the improvement of the computational capabilities of computers and the emerging of various powerful input devices, the sketching techniques are developing rapidly in recent years. There have been some commercial modeling software and packages supporting freehand sketches, examples of which are SketchUp [6], Archipelis Designer [1], and Sunny 3D [5].

Figure 1.3: Sketch-based 3D modeling.

The design of interactive and powerful sketching interfaces requires the exploration of various modeling algorithms and the combination of the knowledge from other fields, such as human psychology, virtual reality, computer vision and so on. At the same time, the sketching technique can also help to make the fundamental modeling and reconstruction algorithms more effective and intelligent. The development of sketch-based modeling systems and the utilization of the sketching technique in improving the modeling and reconstruction algorithms compose the main content of this research.

1.2 Problem statement and objectives

Though a lot of efforts have been made in sketch-based modeling and reconstruction, there are still many important open problems. Sketch-based modeling is often related to human shape perception and human shape perception of 2D sketches is a complicated issue involving various subjects. While sketch-based techniques offer an improvement over traditional methods in terms of accessibility, they are not yet complete replacements. There remains much work to be done for sketch-based modeling to produce a wide range of objects with

high complexity. The challenges for sketch-based modeling include the development of a natural and friendly user interface, the exploration on perceptually meaningful description of a 3D shape, the proper interpretation of the user sketches, and the enhancement of the sketch precisions and model qualities. Meanwhile, it is non-trivial to incorporate sketching techniques into fundamental modeling and reconstruction algorithms in order to produce a faithful and desirable results to the user.

This thesis researches the use of sketching techniques in 3D modeling and reconstruction to enhance various geometric modeling algorithms. It aims to gain deep understanding of diverse techniques from geometric modeling, computer graphics, human computer interaction required to advance the state of the art in interactive 3D modeling, to investigate how sketching interfaces can be efficiently used for enhancing 3D modeling and reconstruction, and to develop sketch-based techniques for creating, editing, and manipulating digital geometric models. In particular, the research objectives are:

- to explore and develop novel sketching interfaces and algorithms for sketch-based modeling, by combining the knowledge of geometric modeling, human computer interaction, compute vision, human psychology and so on;

- to investigate how to enhance the fundamental 3D modeling and reconstruction algorithms by using sketching techniques, thus producing faithful and satisfactory modeling results to the user and making the algorithms suitable for various interactive geometric processing applications.

1.3 Contributions and thesis organization

To accomplish these objectives, the following approaches have been proposed:

First, we developed a sketch-based modeling system with auxiliary planes as references for 3D freeform shape design. The user can first create an initial 3D shape with arbitrary topology by sketching the cross section curves of the model, and then use sketches to per-

form operations such as deformation, extrusion and so on to further edit the model. To regularize and interpret the user's sketches properly, we introduce some rules for the input strokes, which are based on both the semantic meaning of the sketched strokes and human psychology. Unlike other sketching systems, all the creation and editing operations in the proposed system are performed with reference to some auxiliary planes that are automatically constructed based on the user sketches or default settings. The use of reference planes provides a heuristic solution to the ambiguity problem of the 2D interface for modeling in 3D space and makes the user input meaningful and intuitive.

Second, it is observed that most previous sketch-based modeling systems mainly focus on providing editing tools to gradually sculpt a 3D model from a rough shape rather than creating it from scratch through iterative sketches. In order to enhance the creation function in sketching-based modeling, we proposed a progressive modeling approach, which allows the user to create a 3D model through iterative sketches, taking both existing sketched curves and the up-to-date reconstructed 3D shape as references to get fully aware of the shape of the sketched model. A novel surface reconstruction algorithm which ensures gradual shape updating during iterative sketching and produces models with a singly connected component is proposed to effectively support this creation operation.

Third, we studied one important editing function in sketch-based modeling: surface deformation. We presented a new surface-based mesh deformation method that performs the computation via an edge-based graph, which allows natural increase of the sampling rate for more accurate shape computation and generates better deformation results. The user is given the flexibility of adjusting the deformation effect between local shape preservation and global smoothness. Moreover, to simulate the deformation behaviors of regions with different materials, we introduce a stiffness property into the deformation model and present an easy and intuitive way for the user to set the material property as well. Thus, our algorithm can produce satisfactory results and make the deformation more flexible.

Fourth, we utilized sketching in solving the problem of surface reconstruction from cross sections with arbitrary orientations and presented a new scheme which constructs a

smooth mesh surface from cross section curves. The scheme is composed of two parts. The first part is a surface reconstruction algorithm that produces a manifold surface interpolating the input cross sections and having regular local shapes. It is the generalization of the algorithm developed in our progressive modeling which is used to handle cross sections with arbitrary orientations. The second part is sketching assisted topology editing, which allows the user to make further adjustment on the local topology of the reconstructed model through simple sketches if the initial shape is not satisfactory. The computation of this editing operation makes use of the intermediate information obtained during the reconstruction process and avoids complicated analysis on the shape of the global surface. Our surface reconstruction scheme has the unique ability to build a manifold surface with only one connected component, and avoids producing spurious or weired local shapes which are often seen in previous methods. The reconstruction process is further accelerated by computing the sub-surfaces in parallel. This surface reconstruction scheme can be used in various applications including sketch-based modeling.

The remainder of this thesis is organized as follows:

- Chapter 2 reviews the related works, which include sketch-based modeling systems, mesh deformation algorithms and surface reconstruction from cross section curves.

- Chapter 3 presents our sketch-based modeling system which takes planes as references for freeform shape design. The interfaces for the progressive creation and editing operations, the rules for interpreting and regularizing the user sketches, as well as the algorithm on the calculation of the reference planes in this system are described in detail.

- Chapter 4 introduces the algorithm on progressive surface reconstruction from orthogonal cross sections, which effectively supports the creation function in our sketching interface.

- Chapter 5 presents the algorithm on mesh deformation , which is one of the most

important editing functions in our system.

- Chapter 6 introduces a new framework for surface reconstruction from cross sections with arbitrary orientations, which includes generalizing the algorithm in Chapter 4 to handle cross sections with arbitrary orientations and using sketching to edit topology of the resulting surface.

- Chapter 7 summarizes the thesis and points out some directions for future work.

Chapter 2

Literature Review

This chapter reviews the state-of-the-art in the related research areas. We begin from the sketch-based modeling, especially the two important modules for producing 3D models: sketching and sculpting. Then we survey the work on mesh deformation and surface reconstruction from cross sections, which are two fundamental geometric algorithms in sketch-based modeling and reconstruction.

2.1 Sketch-based modeling system

In recent years, a lot of papers on sketch-based modeling have been published [123], proposing different kinds of user interfaces and algorithms. Some of these works make use of existing template models or images to help interpret the user input and reconstruct the target models, while others build 3D models purely from the 2D user sketches without using any prior information.

The template-based systems are characterized by the fact that they have some "memory"of 3D shapes built in, which guide their interpretation of the input sketches. This requires the analysis of the geometric or semantic features of the template shapes, such as the silhouette of the 3D shape under some views, the curvature distributions, etc. These methods are usually dedicated to some specific modeling applications, such as the model-

ing of clothes [152], trees [72, 120, 33], hair [161], clouds [162], architectural shapes [6] and so on.

Contrarily, the other systems take only the user sketches as input and build 3D models from scratch. In such systems, sketching and sculpting, which are the main techniques used by humans to communicate about shapes [30], are adopted for the creation and editing of the 3D models. In the sketching method, the user sketches the 2D curves from scratch and the 3D models with the depicted shape will be constructed automatically. While in sculpting, the user starts the creation from a very rough 3D shape and uses various sketching tools to carve, deform, extend or smooth it to a delicate one. Here we first review the systems using the sketching technique, and then examine various sculpting tools in detail.

2.1.1 Sketching methods

The sketching technique usually lets the user sketch the silhouettes and feature lines to depict the 3D shape and then constructs the model automatically.

Single view sketching: One popular sketching method is to first allow the user to complete complex sketches on the 2D plane under a fixed viewpoint, then interpret and map the 2D curves or strokes into 3D space according to some rules, and finally reconstruct the 3D model corresponding to the input sketches. This is quite similar to people's drawing behavior when sketching an object on a paper with a pen. The way of the interpretations usually conforms to the 10 visual rules proposed by Hoffman [66] for the understanding of the 2D drawings which depict a 3D object in visual intelligence.

The pioneer work Teddy [71] allows the user to sketch a simple closed contour to depict a model with balloon-like shape which takes the curve as its silhouette. This is consistent with the visual rule in [66] that people tend to interpret a curve as the rim of a 3D surface which is as smooth as possible. Then the closed area bounded by the curve is triangulated and the triangle vertices are elevated with the extracted chordal axes to construct a balloon shaped 3D mesh. Similar methods can also be found in [69, 70]. FiberMesh [115] then

improved this method to get a mesh surface with higher quality and smoother shape, by using the method similar to [116] to get the initial triangles with more regular shapes and minimizing two quadratic energy functions to iteratively optimize the mesh surface.

Olga Karpenko et al. [82] proposed a similar system to Teddy, but they used the RBF(Radial Basis Function) implicit surface instead of mesh as the surface representation. By taking the points on the user sketched curve as the "zero points"which correspond to the zero value of the basis function, and generating some extra points which indicate the outside and inside information of the target surface, the parameters of the implicit surface can be obtained. Similar methods can be found in [43, 10, 148, 137, 135, 162, 29]. The implicit surface is naturally smooth and no extra surface optimization is needed. Meanwhile, some further editing such as the merging of multiple surface components can be easily implemented. However, spurious local shapes may be produced due to the intrinsic property of the implicit surface, and the troublesome conversion to triangular mesh is also inevitable.

To build surfaces with higher quality, Nasri et al. [114] proposed a method of constructing a subdivision surface from a closed 2D contour. By making use of the polygonal complex approach [112], it is able to build the control mesh of a Catmull-Clark subdivision surface with simple topology and the limit of the surface will interpolate the input contour. Cherlin et al. [35] presented a system which could generate some special parametric surfaces from the sketched contour, such as rotational blending surfaces and cross section blending surfaces, by extracting the skeleton of the input curve and taking it as the axis of the target revolution surface. The time for constructing objects with relatively complex shapes is usually quite long, making the system unsuitable for interactive modeling applications.

Meanwhile, all these methods are only able to build a smooth object with a simple shape, while the creation of the surface features in these systems mainly relies on the subsequent editing operations.

To produce a complex 3D model with features on its surface through sketching, more curves need to be sketched and more rules should be proposed to interpret these input

curves.

The information of intersection of strokes usually helps to interpret the sketches and reconstruct the models. For example, CrossSketch [12] proposed a method which is able to construct a surface patch with details on it from a few strokes. This approach detects the intersections of the input strokes and makes use of the Cubic Corners method [128] to infer the 3D vertex positions. The 3D surface patch and details which are consistent with the 2D input strokes under the viewpoint of sketching can then be reconstructed sequentially. However, it is only able to construct an open surface rather than a closed 3D mesh, and the Cubic Corners method is not efficient for inaccurate drawings. SmoothSketch [81] used the algorithm proposed in [160] to detect the T-junction points from user sketched contours and infer the hidden contours on the target surface. A smooth surface with balloon-like shape and T-junction points under the current viewpoint can then be reconstructed. A similar method can be found in [40].

Different from the free-form surfaces, the CAD models are characterized by the hard edges, rigid corners and planar faces [123]. The reconstruction of such models relies on the interpretations of the sketched straight lines. These interpretations are also consistent with those proposed in [66] for the understanding of the straight lines of human being's.

The most important step for the reconstruction of CAD models is the identification of the vertices, edges and corners from the 2D user sketches. Early works usually classify line segments from images [105] or let the user to specify them manually [130], which seem tedious for interactive applications. Later approaches improved it by either implementing the reconstruction incrementally with each input sketch [39], or using some rules or templates to create the model after all the input sketches complete [52, 153]. However, these systems are limited to construct objects consisting of only straight lines, and no curves are allowed.

Varley et al. [154] gave an initial attempt on including curves into the sketch-based modeling of CAD models. First of all straight lines are required to create a basic frame of the model, then the user is allowed to give some further sketches to modify the shape of the model and let it contain curves. Masry et al. [107] then made this process fully automatic.

First they selected a vertex with three attached lines as the origin and used the optimization method proposed in [78] to reconstruct the depth of all the endpoint vertices of all strokes. Similar algorithm was also used in [38]. Then the gradient steepest descent algorithm was used to reconstruct the depth of the curves. This method was further improved in [94], which combines the optimization and the Cubic Corners methods to balance between the correct convergence and the efficiency for inaccurate input sketches.

Although the single view sketching methods provide a natural and simple way for the user to create a 3D model by making use of various rules to map the 2D user sketches into 3D, ambiguities on the interpretation still exist when the sketches become complex. Meanwhile, these methods also propose various requirements on the user sketches to guarantee the validity of the input, which limit the user's creativity and also the shape of the reconstructed object to some extent.

Multi-view sketching: To avoid the ambiguity problem in the single view sketching systems and allow the user to give more descriptions on the target shape, some methods have been proposed to let the user sketch the outlines of the model from multiple viewpoints.

Kang et al. [83] presented a system which allowed the user to sketch 2D curves under two different viewpoints and used the Epipolar geometry to combine them into a 3D curve, such that more precise description of a target 3D curve can be obtained. Though being a little troublesome, this method provides a feasible way to define the shape of the target model through sketching multiple curves.

Das et al. [89] allowed the user to interweave the sketching and viewpoint adjustment to build surface patches from sketched boundary curves. Similar method was adopted in [79, 80, 21]. Under each viewpoint, the input 2D curve will be mapped to a 3D curve which has the minimum normal curvature among all the potential 3D curves whose projections overlap with the 2D curve. Thus, the 3D curve will be as smooth as possible. After that, the bounding curves of each surface patch are identified manually, and the final surface can be obtained by stitching these patches. Abbasinejad et al. [8] further made the identification of the bounding curves of each patch fully automatic. The patch-based method was extended

in [15] and [16] which employed a multi-touch display and defined multiple gestures to make the user interface more intuitive and flexible. [125] and [133] further made use of these approaches to build subdivision and developable surfaces respectively. These patch-based methods work well on producing non-manifold surfaces, while for manifold surfaces, the reconstruction will be more involved.

Sowel et al. [146] proposed a system allowing the user to manipulate a virtual view plane to intersect with the given 3D medical data and sketch cross sections along the contours on each plane. Then the algorithm proposed in [101] was used to reconstruct a closed mesh surface which interpolates the non-parallel input curves. This method relies on the predefined volume data, while for sketch-based free-form modeling applications, the input strokes may not be regular, and the analysis and process of them will thus become complicated.

More recently, Rivers et al. [132] presented a system which has a similar interface to the traditional modeling software. The user sketches the silhouettes of a 3D object from three orthogonal views and the final model is computed by using a method similar to the visual hull approach [92]. This method requires much imagination for novice users when a model with multiple components and different depths are desired.

Sketching a 3D model from multiple viewpoints gives more descriptions of the target shape and avoids the ambiguity problem in single view sketching to a large extent. Nevertheless, it is not easy to maintain the coherence and validity of the sketches from different viewpoints, especially for shapes with complex topologies.

Despite all the above mentioned attempts on reconstructing 3D objects from 2D sketches, the modeling result can only be obtained after all the curves have been sketched, thus little hints of the 3D shape are provided to the user during the sketching process. Meanwhile, sometimes it is still difficult to create a 3D object with many details in the sketching methods. The underlying reason is that sketching does not directly communicate the shape people have in mind, and such drawings take a great deal of skill which restricts the medium to trained and experienced designers [30]. Therefore, some subsequent editing operations,

which are similar to the sculpting behavior in real life, are also important for further modification of the initially reconstructed model in sketch-based modeling systems. Next we will introduce the sculpting operations.

2.1.2 Sculpting methods

The sculpting method provides various tools to allow the user to modify the shape of the model and add features on the surface through iterative 2D sketches and gestures. Since the editing operations are implemented directly on the model surface, perception ambiguities on the input sketches and output shapes can be avoided to a large extent. Therefore, the sculpting methods are quite popular and compose a large body of the works on sketch-based modeling.

The sculpting technique can be thought of as the melding of two parts: the original surface and the sketched features. According to the scale of the features, the sculpting operations can be divided into three categories: The big scale feature creation which is used to modify the global shape of the model; The small scale surface augmentation which focuses on creating or erasing detailed features on local parts of the surface; The surface deformation technique which produces surfaces with appealing shapes without introducing any additional features – all elements in the final surface come from the original surface.

To add a surface part with big scale to the existing surface, Teddy [71] provided an extrusion operation to define a new part connecting the original model. It is a two-stroke operation: a closed stroke sketched on the surface suggesting the part to extrude and a stroke depicting the silhouette of the extruded surface sketched on the view plane perpendicular to the former stroke. Then the closed stroke will be swept along the skeleton of the silhouette stroke to construct the extruded 3D shape. Similar functions can be found in [115, 111, 155].

To give a more detailed description to the shape of the part to add, Karpenko et al. [82] proposed to allow the user to first sketch the silhouettes of the new part and then merge

the two parts by sketching a guidance stroke. The use of the implicit surface as the surface representation makes the computation much easier. This approach was also used in [43, 10, 148, 137, 136, 162]. [155, 132, 102] further extended this operation to mesh surfaces by using the CSG-based methods.

To remove a surface part, the cutting [71, 115, 111, 102] and tunneling [71, 115, 137, 136] techniques are often used. The former allows the user to cut a surface part off by sketching the cutting boundary, which is a simplified version of the operation usually seen in interactive mesh segmentation [140]. The latter is used to dig a hole by sketching two closed circles on the frontal and back sides of the model.

The systems [117, 122, 137, 115] provided functions for the user to create sharp or smooth features with small scales on the current mesh. The user drawn stroke is first projected onto the existing mesh surface, then the features are created along the projected stroke by moving the related vertices. Furthermore, the projected sketch stroke can be seen as new positional constraints for some advanced surface optimization operations [115, 48]. Yotam et al. [61] enhanced the feature creation function by using the shading information. It allows the user to sketch shadings on a 3D shape under the current viewpoint to create different kinds of features, just like what the artists do when drawing 2D pictures. Instead of creating the details from scratch, Zhang et al. [169] allowed the user to sketch around the existing features on the mesh surface to enhance them, by making use of a variation of the bilateral filter [73, 53].

Due to the inaccuracy of the user sketches and iterative operations during the modeling process, some local parts on the model may not be smooth. The systems [71, 115, 102] enable the user to sketch back and forth to smooth these local parts, by using various mesh smoothing algorithms.

Note that all the above sculpting operations aim at changing the features on the original surface, regardless of the scale of the modification. To modify the shape of the model while preserving the original features, the deformation tool, which is a very effective and popular function in surface sculpting, is often used.

Existing deformation methods can be divided into two categories: the space-based deformation [56] and the surface-based deformation [28, 163]. The former methods indirectly reshape an object by warping its surrounding space. They are computationally efficient and applicable to a variety of object representations. Since manipulating ambient space directly is infeasible, deformations are controlled by tools of various dimensions, such as points [26], curves [143, 127] and volumes [139, 75, 98]. However in these methods, when large scale deformation happens, the geometric details on the original surface could not be well preserved. Besides, it is not suitable for applications where direct manipulations on the mesh surfaces are needed.

The surface-based method allows the user to deform the mesh by manipulating a handle on it. [115, 145, 144, 147] proposed to let the user drag a point on the surface along the direction parallel to the screen and then compute its new position according to the position of the camera. The new positions of the surface part to deform is then computed according to the positional change of the handle point by solving an optimization problem. This approach can give real-time feedback of the shape change to the user and is suitable for interactive editing, though it is not intuitive enough for confirming the direction of deformation in 3D space and the degree of freedom on the manipulation is limited.

To manipulate multiple handle points at the same time, Nealen et al [117] proposed to deform a mesh by changing the shape of a handle curve on the surface. First, a curve is drawn on the mesh to serve as the handle curve. Then the model is rotated manually to a proper angle and the deformed curve is drawn on the screen. The 3D position of the new curve could be calculated according to the current viewpoint and the deformed mesh can be computed. SilSketch [173] extends this idea by automatically detecting the handle curve which is a part of the silhouette of the model and corresponds to a user input stroke for the deformed handle curve. A similar method can be found in [90]. Since these methods rely on the 2D screen and lack references for mapping the 2D information into 3D, the handle curve and its deformed shape are limited to be planar.

The details of the algorithms on the surface deformation will be introduced in Sec-

tion 2.2.

Different from the sketching technique, the sculpting tools allow the user to directly interact with the model surface. However when the initial shape is far from desired, iterative and boring sculpting operations need to be done. Meanwhile, it will not be easy to preserve the quality of the final shape if too many operations are implemented.

2.2 Surface-based mesh deformation

Deformation is an important tool for the editing of a created model. The deformation methods for parametric or subdivision surfaces have already been developed for years, but these algorithms are difficult to be applied directly to meshes. Due to its importance in various kinds of applications such as geometric modeling and computer animation, mesh deformation has become a popular research topic in digital geometry processing.

As introduced in Section 2.1.2, the surface-based deformation method allows the user to directly manipulate the mesh surface and is thus more suitable for interactive applications than the space-based method which deforms a mesh by indirectly manipulating its surrounding space.

The general goal of the surface deformation is to preserve the local details on the original mesh while keeping the global surface as smooth as possible. To achieve this goal, the mesh is usually encoded into some kinds of representations which reflect the geometric properties of the surface. The deformed mesh will then be calculated by preserving these representations as much as possible. Based on the way of the encoding, surface-based deformation can be further classified into multi-resolution and single-resolution methods.

The multi-resolution methods [174, 62, 87, 88, 93, 25, 106] encode the mesh into a base layer and several levels of refinement, according to its surface geometric properties. The base layer is the low-frequency component which represent the rough shape of the surface and is typically represented in Cartesian coordinates. The refinement are described locally to represent the geometric details of the surface. During the deformation process, only the

low-level layers are deformed and the high levels with more details keep fixed, such that the details of the mesh will be preserved. Using this representation, deformation can be performed on an appropriate user-specified level-of-detail. The multi-resolution approach is the straightforward representation of the details of the shape in a manner that is invariant to the global coordinate system. However, it requires the user to manually set both the base mesh and levels of refinement for meshes with complex details, which seems quite tedious for interactive applications which require fast and convenient operations.

Different from the multi-resolution method, the single-resolution approach encodes the geometric details of a mesh in some measurements in a uniform manner. Since these measurements are computed by local operators, they naturally contain the local relationship between primitives (usually be vertices) and their one-ring neighbors. By preserving these measurements, the mesh details can be preserved and the global shape will be smooth. The user can thus easily implement the deformation by directly manipulating the mesh surfaces and no extra manual intervention is required. These advantages make the single-resolution method a natural choice for the interactive geometric editing applications, such as the sketch-based modeling.

The way of representing the mesh surface in the single-resolution approach comes from that of representing a smooth surface. It is known from differential geometry [31] that the first and second fundamental forms are often used to measure geometric properties of a smooth surface, such as lengths, areas, and curvatures. By minimizing the changes of the two fundamental forms and adding positional constraints for the corresponding surface parts, the geometric properties of the original surface can be preserved during a deformation process, leading to the minimization of the famous thin shell energy [150]. This energy is physically accurate to simulate the deformation of the surface of a real-world object made of physical material. Since the solving of this nonlinear minimization problem is computationally expensive, it can be further simplified by replacing the changes of first and second fundamental forms with the first and second order partial derivatives of the displacement function w.r.t. each point on the surface [32, 158].

For a mesh surface which can be regarded as a piecewise linear surface, different methods are adopted to first represent the geometric properties corresponding to those of a smooth surface. To perform the deformation, the new positions of some vertices are designated and the remaining vertex positions are solved by fitting the new geometric properties of the deformed mesh to those of the original mesh. The vertices whose positions are to be specified usually include the *handle* vertices, which the users want to move to designated new 3D locations and the *static* vertices, which are expected to stay fixed during the deformation process. The other vertices whose positions need to be solved are usually called the *Region of Interest* (ROI) vertices.

We next divide these methods into two categories: the Laplacian-based methods and the other methods. The former formulate the deformation problem based on the Laplacian coordinates, which represent the second order discrete differential quantities of the mesh, while the latter try to solve the problem by encoding the mesh surface into other representations and preserving them.

2.2.1 Laplacian-based methods

The most popular methods to approximate the deformation energy on a mesh surface are the Laplacian coordinates methods, which used the Laplacian vectors defined on the mesh surface to approximate the second order differential quantities of the smooth surface. These methods comprise a large body of work on mesh deformation over the recent years.

Specifically, a Laplacian vector at a vertex is defined as a vector pointing from the weighted center of its neighboring vertices to the vertex itself. Its magnitude and direction approximate the mean curvature and normal direction at that vertex respectively. The cotangent weights scheme proposed in [108] is often used for computing the weights for each neighboring vertex. Usually, the weights are calculated on the original mesh and treated as constants during the deformation, and the vertex positions of the deformed mesh can be calculated by solving a linear system. This method was first used by Marc Alexa in [9].

However, the Laplacian vector is a 3D vector that is not invariant under rotation and scale transformations. So if a deformation that contains rotation or scale happens on a given mesh, preserving the orientation and magnitude of the Laplacian vectors w.r.t. the global coordinate system will cause the distortion of the local details. Thus the transformations of the Laplacian vectors during the deformation process are also considered in subsequent works. These works can be classified into the linear and nonlinear methods.

The linear methods represent the transformation of the Laplacian vector as a linear one, such that the solving of the result system can be easy and fast. Depending on the way of estimating this transformation, the linear methods can be further classified into explicit and implicit types. The former methods specify the transformation explicitly from existing information, while the latter methods evaluate the transformation implicitly from the deformed surface.

Yu et al. [166] proposed an explicit method of propagating the transformation of the handle vertices to the other vertices of the mesh using geodesic interpolation. Besides the positional constraints, the user has to specify the transformation matrices for the handle vertices explicitly. The transformation matrix is then decomposed into rotation and scaling matrices and interpolated over the ROI vertices of the mesh according to their geodesic distances to the handles.

This method is also used in [171], which constructs a graph representing the volume inside the input mesh and encodes the volumetric details by computing the Laplacian coordinates on the graph. By preserving these Laplacian vectors, the unnatural changes in volume can be prevented during the deformation process. However, when more vertices and details are involved in the deformation, constructing the volume inside the mesh and solving the resulting system may become quite time-consuming.

Instead of using geodesic interpolation, Zayer et al. [167] proposed to use the harmonic interpolation, which computes a smooth harmonic scalar filed on the mesh to control the propagation of the transformation. This method shows smoother distributed local transformations and produces better result than the geodesic propagation method.

Popa et al. [129] further used a scalar field to control the weight in the harmonic propagation. The scalar filed which is specified by the user or learned from examples can be used to specify the stiffness of the material of the object, and the deformation of objects with different materials can thus be simulated.

All these explicit methods require manual specification of the transformation at the handles by the user. Moreover, since only rotation and scaling are considered in the transformation, the change of orientation caused by the pure translation of the handle vertices cannot be obtained. This will lead to the distortion of the local shape features.

Different from the explicit methods, the implicit methods try to estimate the transformation automatically from the unknown vertex positions of the deformed surface.

Lipman et al. [99] proposed to estimate the local rotation of the Laplacian vector for each vertex from the original mesh and an initially deformed result calculated using the method in [9]. Then the Laplacian vectors are rotated and a new deformed mesh is calculated using these rotated Laplacian vectors. This process is repeated for several iterations until satisfactory result is obtained. This method of estimating the rotation only works well for relatively smooth surfaces with no largely protruding features. Besides, since scale is not considered in the transformation, this approach cannot preserve the details well under stretching or shrinking.

Sorkine et al. [145] proposed to explicitly represent the transformation of the Laplacian vectors with a linearized matrix which is a combination of the initial and deformed vertex positions, and the deformed mesh can be calculated by solving a linear system. Nealen et al. [117] later used this method in a sketching interface. By adjusting the weights on the preservation of the Laplacian vectors and positional constraints, various deformation results can be obtained. This method works well for deformations with small rotations. However, since rotation in 3D space is inherently nonlinear, the linearization of the transformation matrix makes the method difficult to handle large rotations in a deformation task.

To overcome this problem, Fu et al. [54] uses a two-step strategy to estimate the local transformation. Instead of limiting the transformation matrix to be linearized, this method

allows it to be any affine-transformation. In the first step, this method directly solves the affine transformation matrix associated at each vertex. In order to obtain visual-pleasant deformation result, in the second step, a similarity transformation of each vertex is derived from the solved affine-transformations of its neighbors and then used to transform the Laplacian coordinates for the subsequent mesh reconstruction. This approach enables larger rotations than [145], but it requires tweaking the relative weighting of local transformation smoothness terms; moreover the formulation of implicit transformations may be ill-conditioned for flat one-rings, in which case an extra perturbation is required.

The linear methods are popular due to the robustness and ease of implementation. However, for the sake of speed and robustness, these methods linearize the inherently nonlinear 3D rotation problem. Thus, the computed result will be suboptimal.

Different from the linear methods, the nonlinear approaches formulate the deformation problem as a nonlinear one and use various methods to iteratively solve it.

Huang et al. [67] considered the skeleton, volume and projection constraints in addition to the positional constraints for the vertices based on the Laplacian coordinates method, and the result system becomes a nonlinear one. Then the Gauss-Newton solver is used to solve the energy function iteratively. To make the convergence more steady and faster, it projects the original energy function onto a control mesh to reduce the number of unknown variables. However, since the mean value coordinates it used for editing the constructed control mesh is not a local scheme, a slight editing of the handle vertices may be propagated to the whole mesh and introduce undesired distortions.

To avoid this limitation, Au et al. [13] proposed to represent the mesh surface by a set of isolines of the scalar harmonic field propagating from the handle vertices. These isolines function as a constraint similar to the skeleton constraint in [67] and the local rigidity of the vertices can then be measured and controlled with respect to the isolines. However, the insertion and deletion of handles become expensive operations in this approach, since recomputation and refactorization of the resulting linear system are needed.

Au et al. [14] extended the Laplacian coordinates into the dual mesh domain. Since

the valence of a vertex in the dual mesh is always three, the shape structure formed by a dual mesh vertex and its 1-ring neighborhood is simple, making the convergence more stable. By keeping the magnitude of each Laplacian vector fixed and iteratively updating the normal direction, the deformed mesh can be calculated. Nevertheless, the convergence in this method is slow and it is even difficult to decide when the iteration should stop since no specific energy to minimize is formulated.

These nonlinear methods try to reach optimal solution of the deformation energy function and avoid the inaccurate calculation in the linear methods. However since the computation is relatively expensive, real-time feedback cannot be given to the user when a large number of vertices are involved in the deformation.

Meanwhile, since the Laplacian vectors are related to the second order differential properties of the mesh surface, methods on preserving them only simulate part of the thin shell deformation energy of smooth surfaces. As a result, the deformation result may not be satisfactory.

2.2.2 Other methods

Besides the Laplacian-based methods, there are also some methods proposed to encode the mesh surface into other representations and preserve them during deformation.

To avoid the rotation-variant property of the Laplacian coordinates, Lipman et al. [100] proposed a frame-based representation which is invariant under rigid transformation. This approach first defines a discrete frame at each vertex and records the relative relationship between frames of neighboring vertices into a matrix. By preserving this relative relationship which is rotation-invariant during the deformation, the frames of the deformed surface can be estimated and the new positions of the deformed surface can be solved. This method can preserve the details for mesh deformation under rigid motions, and it was further improved in [97] to handle rotations larger than 2π. This method was also used in the animation editing in [164], in which a linear least-square deformation solver is used to solve the

transformation and vertex positions until convergence. However, since the estimation of the local frames is separated from the positional constraints, rotations caused by the translation of the handles cannot be detected, making this method translation-insensitive.

Alla Sheffer et al. [141] introduced the pyramid coordinates to describe the local shape properties. A projection plane for a vertex and its neighbors is first defined, and the edges between the vertex and the neighbors are then projected onto the plane. The local shape is represented by the angles between the projected edges, the angles between the normal and the edges, and the projected edge lengths. Given several fixed vertex coordinates as boundary conditions, the deformed mesh can be obtained by preserving the initially calculated pyramid coordinates, and the local details on the original mesh can be preserved. However, the computation of this method is too expensive.

Different from the Laplacian coordinates methods which tried to preserve the second order differential properties of the mesh surface, Sorkine et al. [144] presented a nonlinear method to preserve the first order differential properties during deformation. Its basic idea is to use the edge vectors at each vertex to represent the first order differential property at that point, and minimize their changes under rotation transformation. Given an initial guess of the deformed vertex positions, the mesh vertex positions and the rotation matrix can be iteratively updated until convergency. This method can preserve the local details under rigid transformations, while the deformed shape usually appears too rigid and the smoothness is not well guaranteed especially when stretching or shrinking occurs.

To better approximate the thin shell deformation energy on triangular meshes, Eigensatz et al. [49] proposed a nonlinear method to minimize both the changes of the first and second order differential properties of the mesh surface. Especially, these two properties are represented as the inner angles of each triangle and the two principal curvatures at each vertex respectively. The minimization of the energy function is then solved iteratively by using the Levenberg-Marquardt algorithm [104]. This method is further extended in [48] to consider the positional, metric and curvature constraints in a deformation process. Although the Levenberg-Marquardt algorithm is faster and obtains better convergence than

the Gauss-Newton method, this algorithm is still quite slow to converge when the number of vertices becomes larger.

Some recent works proposed to use the analyze-and-edit strategy to preserve the high-level properties of a surface, such as the global structure and specific features. For example, Zhou et al. [172] proposed to first extract the ridge and valley information represented as the principal curvatures [65], and then deform the mesh by using the method in [49], such that the feature lines of the mesh surface can be well preserved. Gal et al. [57] proposed to extract groups of lines to represent the shape of the man-made object. By analyzing and maintaining the individual and mutual relationship of these lines during the deformation, the global structure of the model can be preserved. Similarly, the work [170] proposed to decompose the model into components and use the volumes surrounding each component to represent the shape of the object. The deformation is then carried out by maintaining the mutual relationship of these volumes, and the shape of the model can be preserved at the component level.

2.3 Surface reconstruction from cross sections

Surface reconstruction from planar cross section curves has been researched for years. It refers to the process of reconstructing a smooth surface interpolating a set of input cross sections, either parallel or with arbitrary orientations. It has wide applications in computer graphics and computer-aided design, such as bio-medical modeling, terrain modeling and sketch-based modeling.

The surface reconstruction problem is similar to the skinning problem of parametric surfaces, for which there have been some solutions proposed [113, 134, 68]. However, due to the constraints of the parametric surfaces, the result surface usually approximates, instead of interpolating the input cross sections when these curves are arbitrary. Meanwhile, the topology of the result surface is usually simple. Thus, it is difficult to apply these methods to the surface reconstruction problem.

Early works on mesh surface reconstruction [85, 55, 36, 84, 58, 156] focus on handling the interpolation of parallel cross sections, for example, each of which lies on one slice of the medical data scanned using CT or other devices. In these works, strategies are proposed to find polygonal tiling between each pair of cross sections on neighboring slices. While for slices containing multiple cross sections, these methods do not work since it will be difficult to establish the correspondence between more than two cross sections on neighboring slices.

To avoid the disadvantages of the global approach, [23, 109, 60, 51, 34] used the triangulation methods to compute a volumetric mesh interpolating cross sections on each pair of planes, such that correspondence between the curves can be established and a surface mesh can be extracted. To avoid the complicated calculation in fitting a volumetric mesh, [19, 17, 121, 86, 18, 76, 20] proposed to project the cross sections from neighboring planes onto a common plane and match the projections from different cross sections. The final surface can then be built by connecting the curves and their projections.

Subsequent works proposed various solutions to handling more complicated cross sections on parallel planes. For examples, the works [63, 41, 151] extended the implicit surface method which is often used in the point cloud fitting problem to fit an implicit function to the input curves, extract the isosurface and convert it to a mesh. Some works also considered the normal [29] or material [157] constraints during the fitting process. However the global fitting method using implicit function may produce werid local shapes due to the sparsity of the cross section data. Tessellation of the isosurface should also be taken care of to make sure the final surface still interpolate the input curves, and the time for computation may be long.

There have also been some attempts made to reconstruct surfaces from non-parallel cross sections. [131, 159, 22] extended the implicit function method to build a surface composed of patches, each of which is bounded by cross sections on two adjacent planes. [126, 42, 24] used a divide-and-conquer scheme which first partitions the whole space into zones bounded by the planes that the cross sections lie on. Then within each zone, Delaunay triangulation is used to compute a sub-surface that connects the cross sections on the faces of the

zones. The triangulation methods provide a feasible approach to build surfaces from non-intersected cross sections, while for those who have intersections or even more complex relationship, it will fail to produce a satisfactory result.

A more recent work [101] adopted the divide-and-conquer scheme and extended the method in [76] to handle more complicated non-parallel cross sections. In each zone, the cross sections are projected onto the medial axis planes to establish the correspondence and sub-surface is built by connecting the cross sections and their projections. The quality of the final surface is then improved through iterative smoothing and refinement. This method has been used in the reconstruction of 3D medical data from contours of scanned 2D images [146]. Although it can handle non-parallel cross sections with more complex relationship, the reconstruction result heavily relies on the geometric agency (i.e., the medial axis plane). When the shape of the zone becomes complex, the agency may become irregular, leading to unexpected topological noise to the result surface.

All these methods on handling the non-parallel cross sections aim at building a single object. They used a standard partition scheme in Computational Geometry to subdivide the 3D space into convex arrangements [24]. However, this partition algorithm may separate the connections of different surface components. As a result, the final object will be composed of multiple disconnected components, while in most cases a one-component surface is desired. Contrarily, [47] proposed an approach to build multiple objects from multi-labeled contours at the same time instead of one single object, by defining rules to distinguish components belonging to different objects. Both self-intersections and inter-object intersections between these components can be avoided. While this is useful in some bio-medical modeling applications such as reconstructing neurons from electron microscopy images, the multi-component problem when building one single object still remains.

Chapter 3

Reference Plane Assisted Sketching Interface for Progressive Shape Design

This chapter presents our reference plane assisted sketching interface for progressive shape design. The underlying algorithms that support and implement the sketching interface are also provided.

3.1 Introduction

It has been introduced in previous chapters that sketch-based modeling allows the user to do 3D modeling by sketching 2D strokes, which provides a convenient and intuitive way over the traditional modeling packages using the WIMP-style (Windows, Icon, Menu, Pointer) interfaces. Sketch-based modeling typically uses freeform curves (or strokes) as the basic modeling metaphor and allows user to produce 3D models using sketching or sculpting methods, which are quite similar to the creation behavior of artists in reality. It is suitable for early conceptual design and idea communication.

The processes in sketch-based modeling can roughly be classified into sketching and sculpting [30], which correspond to two modeling phases: initial overall shape creation and shape editing, respectively. In the sketching phase, the user draws strokes as the contours

of a desired shape and an overall 3D model is created after the sketching. In the sculpting phase, the user uses various sketches to perform carving, deformation, extrusion or smoothing to interactively edit the 3D model.

As introduced in Section 2.1.1, many sketch-based modeling systems heavily rely on sculpting and their sketching phase is rather simple. A simple sketching directly creates a very rough overall model, starting from which the sculpting is applied to perform various sophisticated shape editing. The advantage of this approach is that the sculpting process is intuitive and the user can see the intermediate modeling results immediately. The disadvantage is that the sculpting is very tedious and time-consuming when the target model is complex geometrically and/or topologically. Also note that in the sculpting process, the initial strokes drawn in the sketching phase, which represent the user's intention about the shape of the target model, will generally be changed or discarded. On the other hand, there are some sketch-based modeling systems that mainly rely on sketching. They let the user sketch some feature lines or silhouettes of the target model and a good 3D model is created after all the sketches are drawn. As pointed out in [30], nevertheless, such sketching operation does not directly communicate about the shape that the user has in his mind. Without the help of visual mediums for describing the 3D shape during drawings, sketching a 3D model takes a great deal of skill and restricts the medium to trained and experienced designers.

Another challenge is that when doing 3D modeling, we have to find indirect ways to interact with the 3D space using the input devices operating on the 2D computer screen. A classical solution to this problem is to use the Arcball [142] when we view a 3D object in a virtual environment. As for the 3D modeling tasks, this problem will become tougher for novice users who are not familiar or sensitive with the 3D space. It is very difficult, if not impossible, to specify the 3D position and orientation when sketching strokes into free space, without the help of references.

Last but not least, although there have been some general rules on perceiving the user sketches of depicting 3D shapes [66], it remains difficult to interpret and map the strokes

that the user sketches for specific modeling tasks. Proper interpretation of sketching is very important because it is related to whether the system correctly understands the user's intention and whether the creation and editing is successfully performed.

In this chapter, we propose methods to solve these problems. First, we enhance the sketching phase by integrating the modeling capability of the conventional sketching and the intuitiveness and instant modeling of the conventional sculpting. Specifically, in the sketching phase, the user iteratively sketch on three sets of orthogonal planes and all the sketches are preserved to define the shape during the sketching, which make the sketching phase enable to create a complicated 3D mesh. The 3D model is progressively created and displayed immediately after each sketch, which makes the sketching phase intuitive as sculpting and the user aware of the 3D shape he sketched.

Second, to properly map the 2D user sketches into 3D space and make the location and orientation of strokes in 3D space meaningful and intuitive, we provide some reference geometry, with reference to which the strokes were sketched. In particular, besides the shape in its current design stage, we also use some auxiliary planes as reference geometry. The auxiliary planes are automatically constructed based on the user's sketches or default settings, and can be further adjusted manually if not satisfactory.

Third, to partially eliminate the ambiguity of understanding the user input sketches, we introduce some rules for regularizing and interpreting them. These rules consider both the semantic meaning of the sketched strokes and human psychology.

Finally, by integrating all these techniques, we present a novel sketching interface for creating and editing 3D models. It is demonstrated that with help of the reference shapes and planes, sketch-based 3D modeling becomes more flexible and easier, thus leading to a more enjoyable touch of the 3D space for users.

The rest of this chapter is organized as follows: Section 3.2 describes our sketching interface. The underlying algorithms that support and implement our sketching interface are explained in Section 3.3. The experimental results and discussions are provided in Section 3.4.

3.2 User interface

Our sketching interface provides both the sketching and sculpting tools for the user to create and edit a 3D model. The former is used for sketching the shape of the model progressively, and the latter, which includes a series of editing functions such as deformation, extrusion and so on, is used to further edit or refine the created model. The sculpting tool is also sketch-based. With this interface, the user can easily create and edit a 3D object in a short period by sketching strokes.

3.2.1 Sketching tool

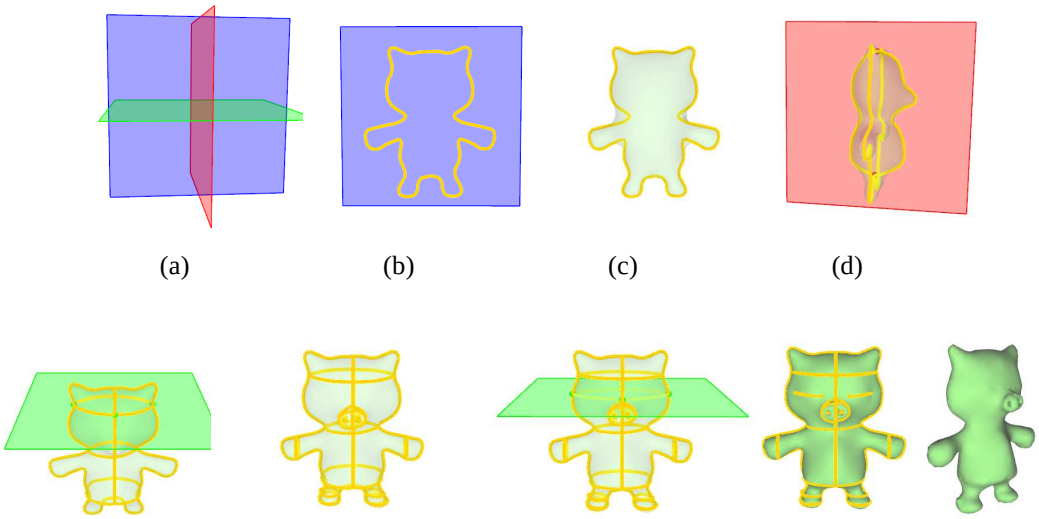


Figure 3.1: An example of using the sketching tool. (a) Initially three basic planes are provided. (b) A closed curve is sketched on the blue plane. (c) The preview of the reconstructed shape. (d) A new cross section is added on the red plane and the previewed shape is rendered. (e) The green plane is translated and three more cross sections are sketched. (f) More cross sections are added to update the overall model. (g) The intersection between the green plane and the current model is rendered semi-transparently to provide reference for further sketches. (i) The final model.

Our sketching tool provides the user three sets of orthogonal planes as reference planes. These reference planes are sketch planes, on which the user can sketch the cross sections

of the model. *Cross section* curves are the intersection curves of the model with a plane. We choose the cross section as the input of the sketching tool for two considerations. First, cross sections are an intuitive and effective description of a 3D model. Figuring out the shape from cross sections is familiar in traditional CAD design and some medical applications. Comparing to silhouettes for shape description, there might exist fewer perception ambiguities using cross sections. Second, a cross section is a planar curve, so it is relatively easy to sketch a cross section with the help of a plane. Meanwhile, taking the cross sections as the input is also consistent with one of the visual rules in [66] that smooth lines are usually interpreted as the contours of the 3D objects by human being.

Initially, the $x = 0$, $y = 0$ and $z = 0$ planes serve as the three basic orthogonal planes (Figures 3.1(a),3.1(d)). The user sketches the cross section of a 3D model on any of these basic planes (Figure 3.1(b)). These three basic planes can be translated along x, y and z axes respectively to allow the sketch of more contour curves (Figures 3.1(e),3.1(f)). This can result in several sets of parallel cross sections to describe the shape of the model. Though we can rotate the basic planes to get the general orientation of the planes, which could provide more flexibilities for the user to sketch the contours, this however makes the algorithm of constructing 3D models from complicated oriented curve networks and design and use of interface much complicated, and increases the occurrence of various ambiguities from sketching. Therefore after a trade-off, we just allow the basic planes to be translated and this is generally sufficient in practice for sketch-based freeform design.

One important feature of our sketching tool is the progressive updating of the shape during sketching. That is, after each sketching, the model that interpolates all the up-to-date sketched strokes will be reconstructed and previewed instantly just as the sculpting process (Figures 3.1(c),3.1(d)), while most existing sketching-based systems create the model only after all the sketches are completed. In general, the sketching tool in the conventional systems is an indirect way of communicating with a shape and it relies on the sense of human perception comparing to the sculpting tool [30]. We provide prompt previewed reconstruction after each sketching, which can help the user be aware of the shape and his

sketching. In addition, the current 3D shape also serves as a reference for the next sketch.

With the up-to-date model available and the current sketching plane, their intersection curve is computed and displayed semi-transparently to provide a reference for the use to sketch on the sketching plane (Figures 3.1(g)). Also during the sketching process, all sketched cross sections are displayed and they can be modified through over-sketching or deleted. In addition, both the 3D model and the auxiliary planes are rendered semi-transparently to avoid blocking the user's view on existing sketches and the whole 3D space when providing sketching references.

Our tool is also able to automatically rotate the view to make the user selected plane for sketching face him as much as possible. This is inspired by the observation in [15] that a good view for the user to sketch 2D strokes is the one that has a large visible projection. With this function, frequent manual viewpoint changes that interrupt the user's attention on sketching can be avoided.

It can be seen from Figure 3.1 that with the help of reference planes and the previewed shapes, more descriptions on the shape of a model can be sketched and the perception ambiguities can be reduced. This minimizes the unnecessary back and forth adjustment of sketches and speeds up the production process of 3D modeling.

Note that our sketching cross sections on a reference plane that intersects the previewed shape looks similar to the sculpting process as in [117, 90], which creates a 3D model by iteratively oversketching its silhouette and deforming it. However, there are intrinsic differences between both processes. First, the cross sections that the user sketched earlier remain unchanged in our sketching process while they usually change with shape deformation in sculpting process. Second, the previewed shape in our process just provides a reference for subsequent sketches. It sets little limitation on the sketches. By contrast, in the sculpting process the sketches are restricted to the current 3D model.

It should be pointed out that even though references have been provided for defining the shape of a model, sometimes ambiguity still occurs. The ambiguity arises from two aspects: the user input strokes may not necessarily follow the same style – different users

may have different manners for defining the shape of a model; even with the same contours, the understandings of the shape they defined may vary among different users. So some rules must be provided for sketching the cross sections. The details of the rules will be discussed in Section 3.3.2.

3.2.2 Sculpting tool

After the user has sketched the overall shape of the model, a set of sculpting functions is provided for further editing and refinement. The sculpting functions include extrusion, tunneling (digging a hole), cutting, smoothing, and deformation, all of which are based on the user's sketches. A stroke recognition mechanism is used to judge the user's intention according to the sketches and switch to the corresponding mode automatically. The sculpting tool accepts three types of strokes: a simple open stroke, a closed stroke, and a scratch-out stroke. The strokes are drawn with reference to the current model. Based on the types of strokes, reference planes may automatically be generated for subsequent sketching.

Basic functions

Next we introduce the types of strokes our system accepts and the corresponding sculpting functions.

- When a closed stroke is sketched on the surface of the model, it triggers an extrusion or tunneling as in [71, 115]. Different from [71, 115], our tool automatically generates a reference plane orthogonal to the closed curve. The plane provides a reference place in 3D space for the user to draw the silhouette curve of the extruded/dug part (Figure 3.2(c)). The orientation of the plane can be further adjusted by performing rotation about one of two axes: the line passing through the center of the closed stroke and orthogonal to the stroke as much as possible or the line connecting the first drawn point of the closed stroke and the farthest point to it on the stroke. The reference plane helps to determine the position of the silhouette curve and gives flexibility to adjust

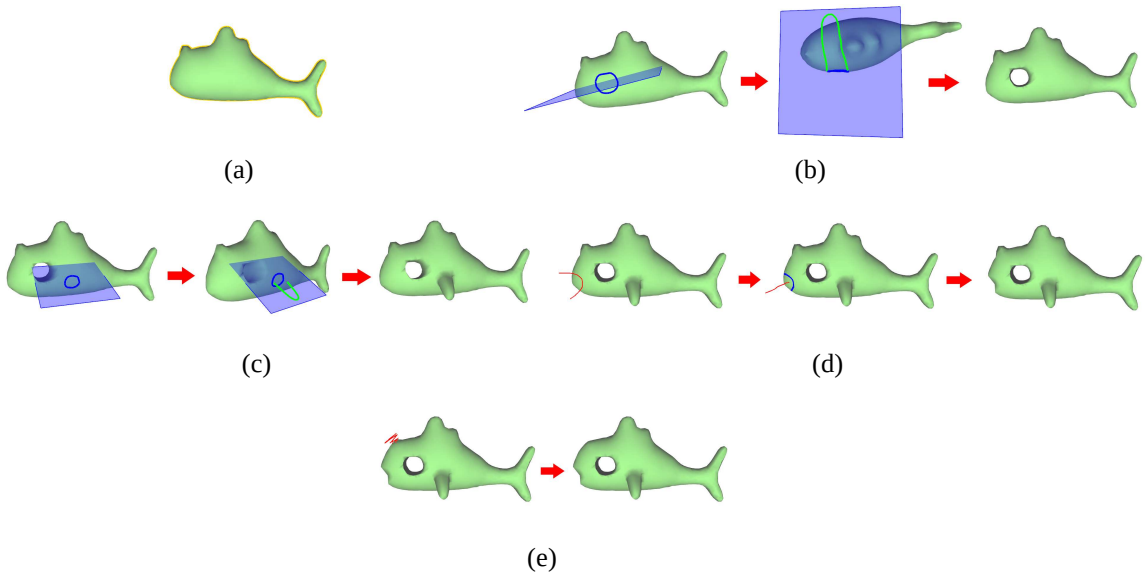


Figure 3.2: Some sculpting examples. (a) A user sketched contour (yellow) and the generated initial model. (b) Tunneling operation. (c) Extrusion operation. (d) Cutting operation. (e) Smoothing operation.

its direction. When the silhouette curve intersects with the backside of the surface, the operation becomes tunneling and a hole is dug (Figure 3.2(b)).

- When the user scratches a part on the surface back and forth, it is regarded as a smoothing operation and the corresponding local part will be smoothed (Figure 3.2(e)).

- When a simple open stroke is sketched, the operation may be cutting or deformation, depending on whether the stroke passes across the model in the image space or not. If it is a cutting stroke, a following stroke will be required to indicate which part to cut off (Figure 3.2(d)).

- If a simple open stroke does not pass across the whole model in the image space, the stroke is mapped to the surface and serves as a handle curve for deformation. The deformation of the surface is driven by the modification of the handle curve. Curves have been shown to be more effective in manipulating the shape deformation [115]. See Figure 3.3 for an example. The subsequent operation of deformation will be introduced next.

Deformation tool

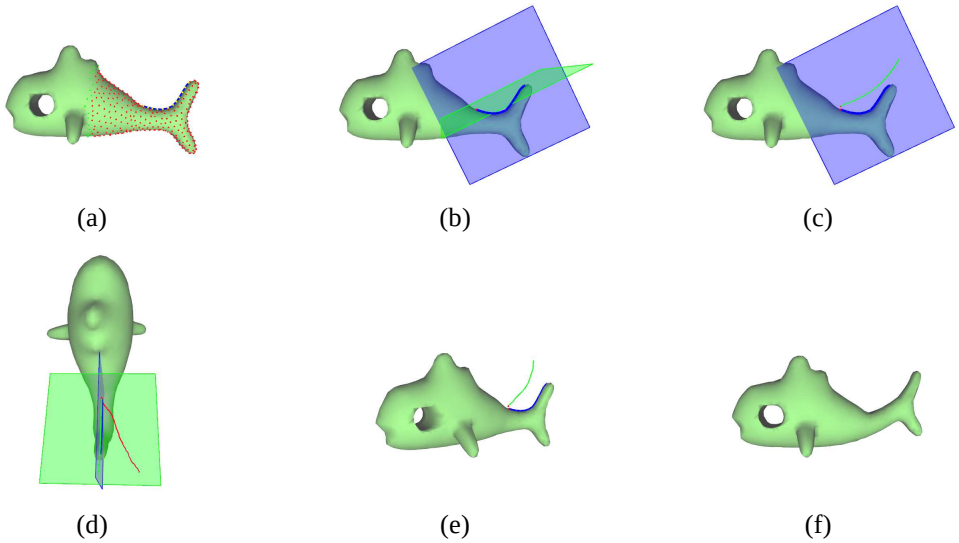


Figure 3.3: An example of deformation. (a) The blue curve is the handle curve sketched on the initial model. The yellow, red and green dots denote the handle, ROI and static vertices respectively. (b) The automatically generated base plane (blue) and projection plane (green). (c) The target curve (green) drawn on the base plane. (d) To create a non-planar target curve, a curve (red) is drawn on the projection plane as its projection. (e) The updated non-planar target curve (green). (f) The deformed model.

The deformation tool is an important sculpting function in our system. As introduced in Section 2.1.2, to deform a mesh, a handle needs to be first specified for the following operations. The handle curve is represented by a polyline consisting of vertices on the surface, called the handle vertices (Figure 3.3(a)). Based on the stroke, two reference planes are generated for the user to sketch the modified shape of the handle curve: a base plane that passes the handle vertices as many as possible for sketching the basic shape of the target curve and a projection plane that is orthogonal to the base plane for drawing the projection of the target curve (see Figure 3.3(b)). In general, the base plane is sufficient for sketching the target shape of the handle curve. However, if the target handle curve is desired to be non-planar, the projection plane is used to sketch the projection of the target curve. The two planar curves on the base plane and the projection plane are combined to determine a a non-planar handle curve (Figure 3.3(c) - Figure 3.3(e)) using an algorithm proposed in [37]. In addition, the two planes can also be rotated to provide more flexibility. In this way, the

user can generate complex 3D target curves.

Further, the user needs to specify the vertices of the mesh within a region of interest (ROI), which will move during the deformation. This is done by three methods. A simple method is to specify a threshold and then to select those vertices on the mesh, whose Euclidean distances to the handles are shorter than the threshold. The second method is to let the user draw a circle on the 2D screen. The ROI vertices are the ones whose projections onto the screen are inside the circle. To have more control in selecting the ROI vertices for complex models, the third method is to use a brush. The vertices brushed are chosen as the ROI vertices. After this, the deformation is performed, which pulls the handle curve to the target location and adjusts the ROI vertices accordingly. The rest vertices remain unchanged and are called the static vertices (see Figure 3.3(a)).

These sculpting tools are all sketch-based and quite convenient to use. Together with the sketching tool, a complete modeling system is built up. With this system, the user could easily create and edit a 3D object in a short period by sketching strokes.

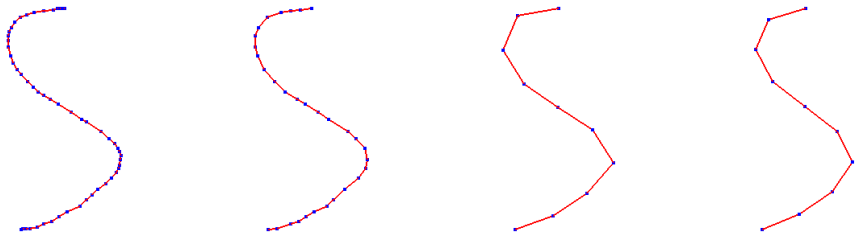
3.3 Algorithms

In this section we describe the underlying algorithms of the sketching interface. We first introduce the preprocessing algorithm for the input strokes. Then we propose the stroke rules for interpreting input cross sections and making them valid for the computation of the 3D model. Finally the algorithm for the calculation of the reference planes in the sculpting tool is provided.

3.3.1 Input stroke preprocessing

For sketch-based modeling systems, the original input stroke created from an input device usually cannot be used directly due to two reasons: 1). The points of the stroke are noisy because of the shaky nature of handling the input devices; 2). the spatial and temporal quantization of the input by the hardware may contain error since the speed that the user

draws is random. Thus, we perform a three-step preprocessing of the input stroke in each functions, as shown in Figure 3.4.



(a) (b) (c) (d)

Figure 3.4: An example of stroke preprocessing. The goal here is to filter the noisy points and get a smooth curve with 10 points. (a) Initial input. (b) Filtering result. (c) Resampling result. (d) Final result after smoothing.

First of all, a point whose distance to its previous one is less than a given threshold will be judged as a redundant noisy point and deleted.

Then we resample the stroke to make its points number a predefined value N_{pre}. To do this, we first calculate the summed distance S_{total} between each two points to get the length of the stroke. Then through dividing S_{total} by N_{pre}, we get the expected unit distance $S_{unit} = S_{total}/N_{pre}$ along the route of the stroke between each two new points. Finally starting from the first point, we "travel"along the original stroke trajectory. Every time when a unit distance S_{unit} reaches, we position a new stroke point. The "journey"ends until the last new point is placed. A new curve is then formed.

This simple resampling algorithm can produce a stroke with a desired number of points. The original shape of the stroke is preserved as much as possible. Meanwhile, it is quite easy to implement and avoids complicated computations required in some existing methods (for example, fitting a polyline to a B-spline and evenly sample it [37]).

Finally the stroke is smoothed by moving each point according to Eq. 3.1.

$$P_i' = \frac{P_{i-1} + 4 * P_i + P_{i+1}}{6} \tag{3.1}$$

where P_i and P_i' are the old and new positions of point i. P_{i-1} and P_{i+1} are the old positions

of its two neighbor points.

The stroke preprocessing is a fundamental step for the functions in the system and helps to produce a visually-pleasant 3D model.

3.3.2 Stroke rules

As mentioned in Section 3.2.1, to avoid the perception ambiguity, some rules are needed to regularize and interpret the user sketched strokes before we construct the surface. It is suggested in the guidelines for the user interface design in [110] that the interface should match with the real world conventions, which is coincident with the user psychology that the user always prefer familiar concepts rather than system-oriented terms [59, 46]. Thus, these stroke rules should make the input curves consistent with the common sense of the cross sections which represent the intersections of the planes and the surface of a 3D object with no self-intersections. Specifically, these rules include:

1. For curves on a plane, if they do not intersect with or contain each other, the interior of each curve belongs to the interior of the model.

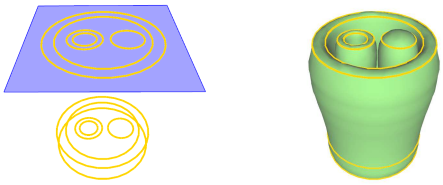
2. For two curves S_1 and S_2 on a plane, if S_1 is totally contained in S_2, then it means that the intersection between the plane and the model is a ring. If the containing relationship exists among more than two curves, that is, we have n curves $S_1, S_2, ..., S_n$ and S_i is within S_{i+1}, then we start the perception from the outermost pair of curves S_n and S_{n-1}.

3. If a curve is self-intersected or intersects with any other curves lying on the same plane, then it is treated as an invalid curve and thus discarded.

4. For two curves S_1 and S_2 lying on two orthogonal planes P_1 and P_2 respectively, the intersection point set $\{V_{S_1P_2}\}$ of curve S_1 and plane P_2, must be exactly the *same* as the set $\{V_{S_2P_1}\}$ of S_2 and plane P_1.

Next we give some explanations of these rules. The first rule is easy to understand. When a user draws one or more closed strokes, the strokes are usually regarded as the outer contours of an object.

When a user tries to make a hole on the mesh surface to create a relatively complex model, a pair of curves are often used, of which one is contained in the other. The ring between the two curves is the interior part of the model. If more holes are desired, more pairs of curves should be used. To avoid the ambiguity of perception, the order of analysis is inward starting from the outermost pair. For example, suppose we have n curves $S_1, S_2, ..., S_n$ and S_{i-1} is totally contained in S_i. Then S_n and S_{n-1} form a ring, S_{n-2} and S_{n-3} form another ring which is contained in the previous one, and so on. If n is an even number, then the ring formed by S_2 and S_1 is innermost; Otherwise we just apply rule 1 to S_1 – the interior of S_1 is the interior of the model. An example of rule 2 is shown in Figure 3.5. Rule 2 makes it possible to model a mesh with arbitrary topology directly by sketching the contours at the creation phase.



(a) (b)

Figure 3.5: An example for rule 2. (a) The user sketched cross sections. Containing relationship exists among multiple cross sections on the blue plane. (b) The generated model.

Rule 3 just does not allow self-intersection of one curve or intersection between any two curves on the same plane since it unlikely happens for the cross section contour of an object to have intersection.

Rule 4 can also be stated in another way that the intersection points between the two curves and the intersection line of the two planes must be be in the same positions. This rule actually conforms to our common understanding of appearance of a 3D object. It is difficult to imagine what the shape of the model will be if the two sets $\{V_{S_1P_2}\}$ and $\{V_{S_2P_1}\}$

are not same. However in practice, it is almost impossible for the users to draw two exactly intersected curves manually. So after the curve S_1 is drawn, we highlight the point set $\{V_{S_1P_2}\}$ to help the users carefully draw the curve S_2 to make it pass through $\{V_{S_1P_2}\}$ (see Figure 4.4(a)). Then we perform a curve deformation procedure, which is explained below, to automatically sew the two curves up before constructing the surface.

For simplicity, let us assume there are two elements in set $\{V_{S_1P_2}\}$, which implies two curves S_1 and S_2 should intersect at two points. So the goal here is to deform the two curves, making them intersect at two points and meanwhile preserve their original shape as much as possible.

First we calculate the intersection point set $\{V_{S_1P_2}\}$ between S_1 and P_2, and $\{V_{S_2P_1}\}$ between S_2 and P_1. Then we find two pairs of different closest points, in each pair the two points are from $\{V_{S_1P_2}\}$ and $\{V_{S_2P_1}\}$ respectively. The midpoints V_{m1} and V_{m2} of these two pairs are treated as the new positions of the handle points. Two handle points on each curve can then be chosen as the points closest to V_{m1} and V_{m2} respectively. Next a range of points near the handles are to be moved with them and we need to compute their new positions. So we could perform the Laplacian deformation algorithm to the two curves to deform them.

Since the two curves are both planar, it can be further simplified into a 2D deformation problem and we used the method in [50] to solve it.

First, we compute the Laplacian vector δ_i at each curve point V_i as:

$$\delta_i = L(V_i) = V_i - \frac{1}{2}\sum_{j \in N(i)} V_j, \tag{3.2}$$

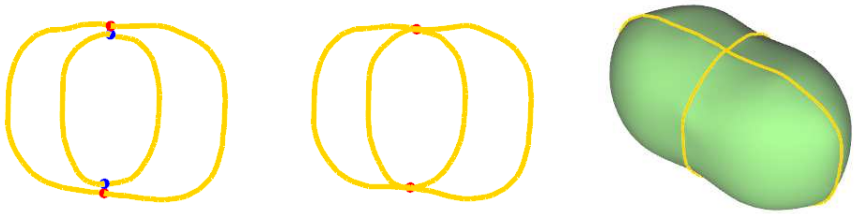
where $\{V_j, j \in N(i)\}$ are the positions of the two neighboring points of V_i. Supposing there are n points involved in the deformation, and considering the positional constraints $\{C_i\}, i \in \{m, ..., n\}, m < n$, the deformation energy we aim to minimize for computing the new point positions is:

$$E(V') = \sum_{i=1}^{n} ||L(V_i') - T_i\delta_i||^2 + \sum_{i=m}^{n} ||V_i' - C_i||^2, \tag{3.3}$$

where T_i is the matrix representing the transformation of δ_i during the deformation process. T_i can be derived by estimating the transformation of the point V_i and its neighbors through minimizing:

$$E(V_i') = ||V_i' - T_i V_i||^2 + \sum_{j \in N(i)} ||V_j' - T_i V_j||^2. \tag{3.4}$$

Since T_i contains 2D rotation and isotropic scale transformations, it can be represented as a skew-symmetric matrix $T_i = \begin{pmatrix} s_i & \omega_i \\ -\omega_i & s_i \end{pmatrix}$. Thus, T_i is a linear combination of V_i' and V_j'. By substituting T_i into Eq. 3.3, the point positions of the deformed curve can be solved.



(a) (b) (c)

Figure 3.6: Curve sewing. (a) Two curves lie on two orthogonal planes. The colored points denote the intersection points between a curve and a plane that another curve lies on. (b) Deform the two curves so that they intersect at two points. (c) Generate a mesh that takes the two curves as contours.

It can be seen from Figure 3.6 that during the deformation process, no extra discontinuity will be caused at the intersection part of each curve and the deformed curves keep their original shapes very well, thus preserving the original characteristics of the user input strokes.

The above mentioned steps are for the simple two-contour case. For multiple contours, we just need to deal with them one by one to make them accord with the rules we defined.

The curve sewing process can handle inconsistent cross sections automatically, which meets the requirement on the user interface design that the system should constructively suggest a solution to help users to recover from errors [110]. By proposing these rules and strategies, our sketching interface will become more natural and intelligent to the user.

After all these processes, we are finally ready to construct the mesh model. The details on the progressive reconstruction algorithm will be presented in Chapter 4.

3.3.3 Generation of reference planes

One problem in the sculpting phase is how to automatically generate reference planes, which have appropriate position and direction for the user to further sketch, once a stroke is drawn with reference to the current model to trigger a deformation, extrusion or tunneling. For a deformation stroke, we propose to let the base plane pass through as many points of the handle curve as possible. Hence the plane can be computed by the least squares fitting. In particular, assume the plane equation is

$$Ax + By + Cz + D = 0, \tag{3.5}$$

where A, B, C and D are the unknown parameters to be determined, and we let $A^2 + B^2 + C^2 = 1$, which means $N = [\ A \quad B \quad C\ ]$ is the unit normal vector of the plane. Then our task is to find the plane that minimizes the following objective function

$$\begin{aligned} f(A,B,C,D) &= \sum_{i \in HC} \frac{(Ax_i + By_i + Cz_i + D)^2}{A^2 + B^2 + C^2}, \\ &= \sum_{i \in HC} (Ax_i + By_i + Cz_i + D)^2, \end{aligned} \tag{3.6}$$

where $i \in HC$ means $p_i = (x_i, y_i, z_i)$ is a point on the handle curve.

To solve this optimization problem, we used the method proposed in [138]. First, we set the partial derivative of $f(A,B,C,D)$ w.r.t. D equal to zero and get:

$$\begin{aligned} \sum_{i \in HC} (Ax_i + By_i + Cz_i + D) &= 0 \\ D &= -(Ax_c + By_c + Cz_c), \end{aligned} \tag{3.7}$$

where (x_c, y_c, z_c) represents the centroid of all the input points on the handle curve. Then we substitute D back to Eq. 3.6, and the function to minimize can be written as:

$$\begin{aligned} f(A,B,C) &= \sum_{i \in HC} (A(x_i - x_c) + B(y_i - y_c) + C(z_i - z_c))^2 \\ &= (NM^T)(MN^T) \\ &= N(M^T M)N^T, \end{aligned} \tag{3.8}$$

where $N = [\ A \quad B \quad C\ ]$ is the unit normal vector of the plane we have mentioned and M is the matrix defined as:

$$\begin{bmatrix} x_1 - x_c & y_1 - y_c & z_1 - z_c \\ x_2 - x_c & y_2 - y_c & z_2 - z_c \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ x_n - x_c & y_n - y_c & z_n - z_c \end{bmatrix},$$

assuming there are n points on the handle curve. Next, $f(A, B, C)$ can be minimized by computing the singular value decomposition of $M = USV^T$ and letting P equal to the column of V that corresponds to the smallest singular value of M. To this end, the parameters of the optimal plane in Eq. 3.5 can be obtained. It can be seen that the calculated plane will interpolate the centroid of the input points and its normal vector is the singular vector of M corresponding to its smallest singular value.

The projection plane is computed from the base plane. First, we project the starting point of the handle curve onto the base plane. Second, we choose another projection point of the handle curve on the base plane, which is farthest to the projection of the starting point. Connecting the two projection points forms a line. Finally, we rotate the base plane about the line by 90 degrees, which gives the projection plane.

After the reference planes are constructed, the user can sketch the new shape and position of the handle curve. If the user intends to draw a non-planar curve by sketching the

shadow of the base plane curve on the projection plane, we use the algorithm proposed in [37] to combine the two planar curves to generate a spatial one.

That is, we first filter the shadow curve and resample it to make the number of points equal to that of the handle curve on the base plane, by using the method proposed in Section 3.3.1. This way, there will be a one-to-one correspondence between each point P_i^b on the planar handle curve and P_i^p on the shadow curve. Then we project P_i^b onto the projection plane to get a projection point P_i^{bp}, and compute the vector $V_i = P_i^p - P_i^{bp}$ which points from P_i^{bp} to P_i^p. The point P_i on the new handle curve can be obtained by translating P_i^b along the vector V_i (i.e. $P_i = P_i^b + V_i$), and the desired handle curve can be formed by connecting each new point P_i sequentially.

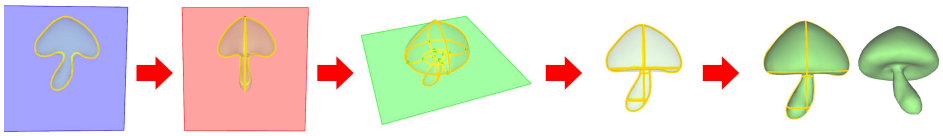
For an extrusion or tunneling stroke, we compute the reference plane in a way similar to computing the projection plane. We first compute an initial least squares plane for the stroke. Then we compute a line on the plane as the rotation axis, which connects two projection points of the stroke. Finally the desired reference plane is obtained by rotating the initially obtained plane about the axis by 90 degrees.

Note that the above methods just give a suggestive orientation of the reference plane. If the user is not satisfied with it, he can rotate the plane about a line which is constructed similarly as in computing the projection plane.

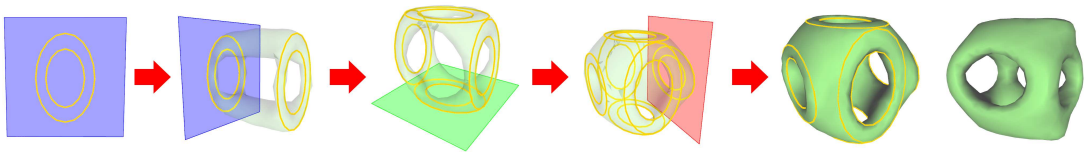
3.4 Results

Our sketch-based modeling is implemented using C++. The experiments are carried out on an Intel Pentium 4 3.6GHz machine with 1G RAM. Both the surface reconstruction and other sculpting functions run in an interactive rate, which meets the requirement for a sketch-based modeling system.

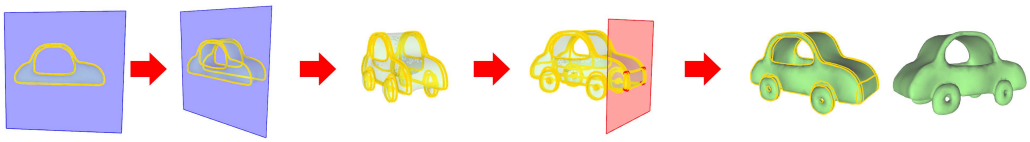
We trained ten novice users for about 20 minutes and then asked them to create some freeform shape models. It was observed that most users tended to use the sketching tool to iteratively sketch and construct the 3D models in most of the time and just took the



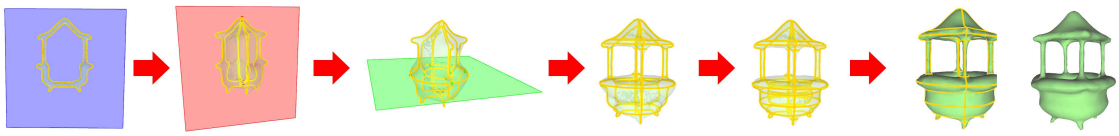
(a) Mushroom



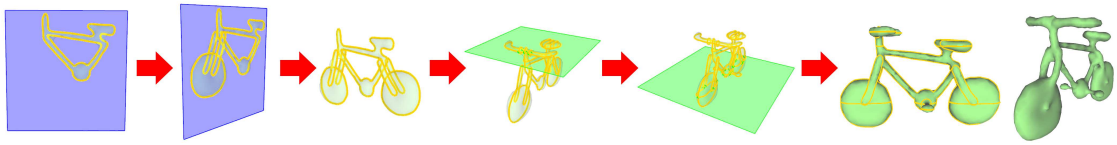
(b) Handicraft



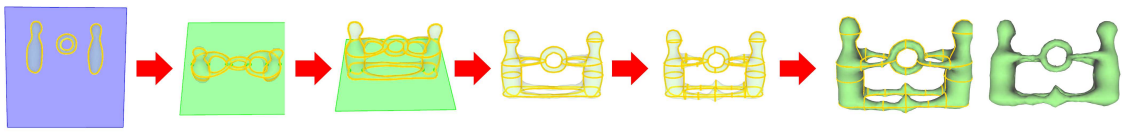
(c) Car



(d) Censer

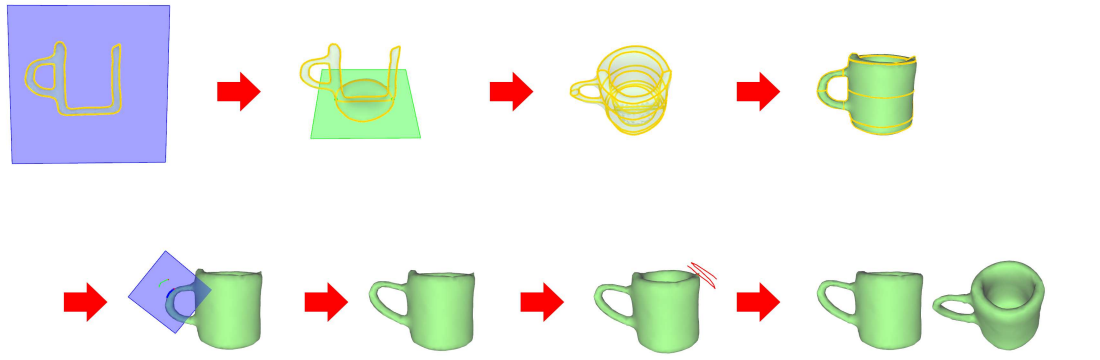


(e) Bicycle

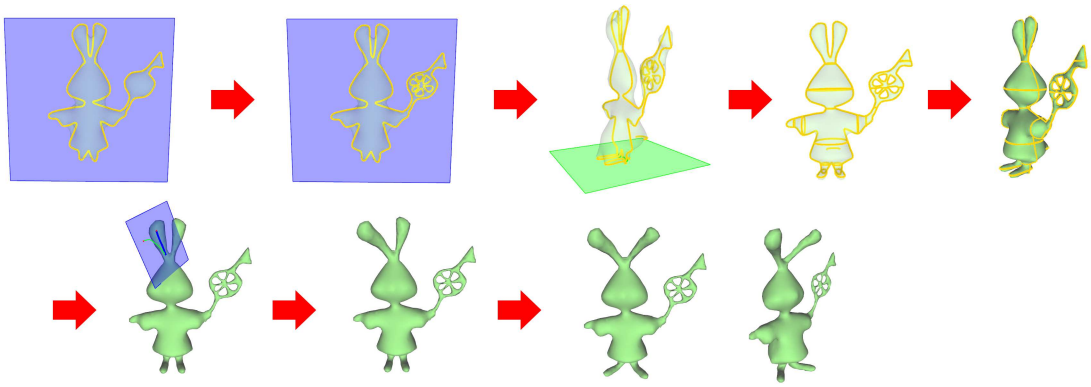


(f) Sculpture

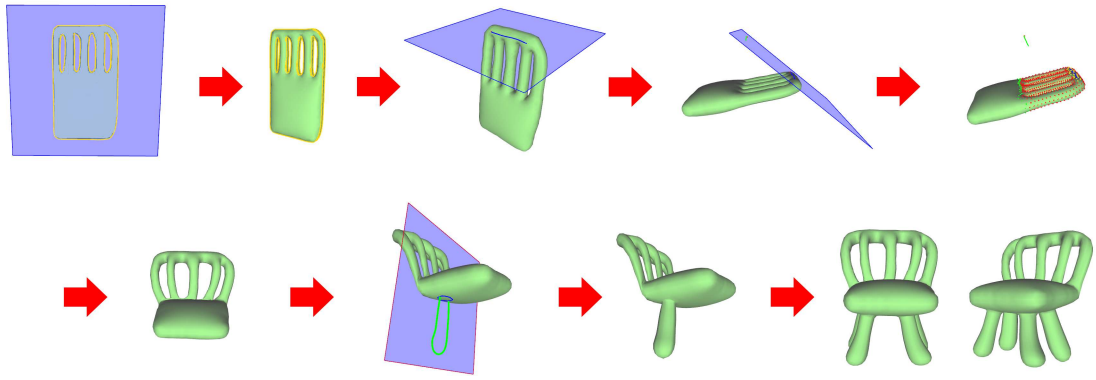
Figure 3.7: Sketching of some models using our system.



(a) Cup



(b) Rabbit



(c) Chair

Figure 3.8: Sketching and sculpting of some models using our system.

sculpting tools to do some detailed modifications and very few preferred to build a model by sculpting from a rough shape. This is consistent with the observation in [30] that the sketching technique is more familiar to public than sculpting. Sculpting requires tedious manual work. Some models created using either the sketching tool or both sketching and sculpting tools are displayed in Figure 3.7 and Figure 3.8.

We learned from the feedback that with the help of the reference planes, the users can get a direct "touch" of the 3D space and they only need to focus on the sketching of the curves for defining the shape of the 3D model. The up-to-date shape together with the existing sketches lets them get a clear idea of the 3D model in each design stage and also gives hints for the next sketches. This can overcome the difficulties on confirming the position and orientation of the 3D curves corresponding to the 2D sketches and imagining the sketched 3D shape.

Chapter 4

Progressive Surface Reconstruction from Orthogonal Cross Sections

This chapter presents the algorithm on progressive surface reconstruction from orthogonal cross sections, which effectively supports the sketching function in our sketch-based modeling system.

4.1 Introduction

Surface reconstruction from cross sections has already been a widely-investigated problem for years. It has wide applications in many areas of computer graphics and computer-aided design. The input is usually a set of planar cross section curves which are extracted from scanned 2D slices or sketched from scratch, and the output is a smooth surface which interpolates the input curves.

The input of the surface reconstruction function is a set of cross sections $CS = \{cs_i \mid i = 1, ..., m\}$ which are progressively sketched by the users and lie on a set of orthogonal planes $P = \{p_i \mid i = 1, ..., n\}$ in 3D space $\mathbb{R}^3$. Each plane p_i can be defined by $x = d_i$, $y = d_i$ or $z = d_i$ ($d_i \in \mathbb{R}$). The cross sections are represented as polylines. In the progressive modeling process, the output in each step is expected to be a closed 2-manifold triangular

mesh M interpolating the sketched cross sections $CS' = \{cs_i \mid i = 1, ..., m', m' \leq m\}$. Each edge of M is incident to two faces and the faces incident to a vertex forms a closed fan.

As has been introduced in Section 2.3, most previous works mainly focused on the reconstruction from parallel cross sections, while the method in [101] is able to build a smooth mesh from cross sections on non-parallel planes and with complicated mutual relationship. It first used the classical algorithm in Computational Geometry to partition the 3D space into zones bounded by the planes that the cross sections lie on. Then within each zone, the cross sections are projected onto the calculated Medial Axis(MA) planes. Next the cross sections and projections are connected to form the initial mesh, whose quality can be improved through iterative refinement and smoothing. This method is designed for applications which only require a final reconstruction result given all the input cross sections, such as bio-medical modeling and so on. However, it cannot meet some requirements in our progressive sketching interface. Providing the cross sections are valid under the stroke rules proposed in Section 3.3.2, these requirements include:

1. The update of the model shape during the progressive modeling process should be natural regardless of the input orders of the sketched curves, and unexpected shape change should be avoided.

2. The reconstructed model is expected to have only one connected component, when the user iteratively adds new sketches that are closely related to the shape in the current design stage.

3. The shape of the final model should be insensitive to the user's sketching order. In other words, given a same set of curves drawn with different orders, the reconstruction result should be unique.

The new surface reconstruction algorithm presented in the rest of this chapter meets the above requirements.

4.2 Overview of the algorithm

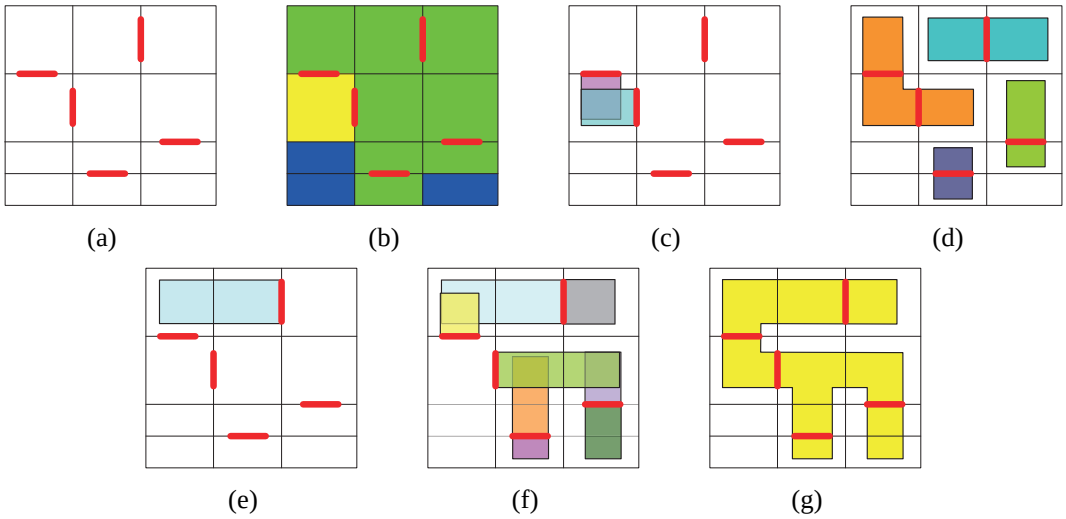


Figure 4.1: 2D illustration of our surface reconstruction algorithm. (a) Input segments (red, corresponding to the 3D cross sections) and partitioned zones. (b) The *empty* zones (blue), *end* zones (green) and *body* zones (yellow). (c) Generating polygons (corresponding to the cylinders) in *body* zones. (d) Generating polygons in the *end* zones using the same way as the reconstruction in the *body* zones. The reconstruction result contains multiple components. (e) Generating extended polygons for an *end* zone. (f) Generating polygons for all *end* zones. (g) The reconstruction result.

Similar to [101], our algorithm also follows the general divide-and-conquer strategy. That is, we first place a large virtual bounding box containing all the cross sections (we set its size to be 2 times that of the bounding box of all points on the cross sections). This box will keep fixed during the sketching process and all points on the input cross sections will lie within it. Then we partition the box into zones $ZN = \{zn_i \mid i = 1, ..., m\}$ using the planes that the cross sections lie on. The shape of each zone zn_i is thus a cuboid since we only allows the translation of the orthogonal reference planes. Within zn_i, the curves might be split into segments such that the segments are entirely lie in the faces $F = \{f_{ij} \mid f_{ij} \in zn_i, j = 1, ..., n\}$. Next we build a sub-surface which interpolates the curve segments on the faces of each zone and stitch all the pieces of sub-surfaces together to form a complete surface. Finally we carry out constrained refinement and smoothing to the initial mesh surface to improve its quality while maintaining its interpolation on the

input cross sections.

Though the projection-based method in [101] can be used for reconstruction, it will make the shape of the reconstructed surface rely on that of the geometric agency (the MA plane) to a large extent. In the progressive modeling process, each time a new cross section is sketched on a new plane, the MA planes in the related zones will be re-calculated, which probably leads to the shape change of the surface components generated from all the cross sections within these zones, and further the change of the sub-surfaces. In that case, some surface parts generated from some unrelated cross sections will be changed, resulting in unexpected shape changes of the model surface. Especially in the iterative sketching process, different input orders of a same set of sketched curves will introduce different changes of the agencies, and the shape changes will be unexpected under some input orders of the curves. In other words, whether gradual shape changes can be achieved or not depends on the sketching orders to some extent. Such an example is shown in Figure 4.2. When new cross sections are added following the order in Figure 4.2(a), unexpected shape change (breaking the originally connected component) happened, which obviously goes against user's intention.

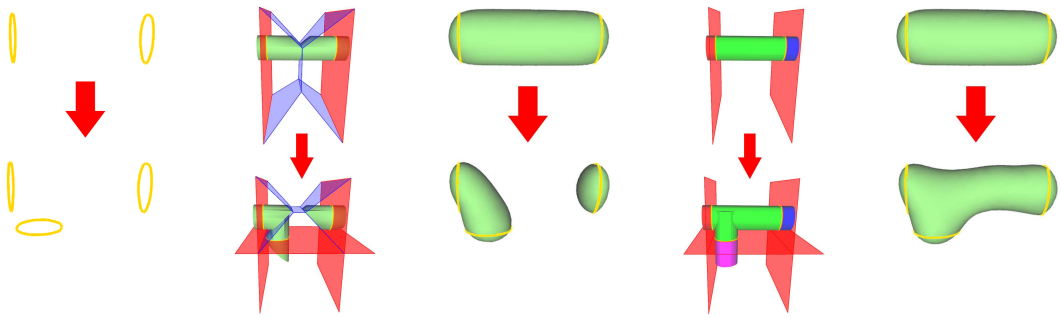


Figure 4.2: An example of comparing our surface reconstruction algorithm and that of [101]. (a) Two cross sections are sketched first and then one more is added. (b) The initial reconstruction results in [101]. The Medial Axis planes (blue) in a zone are shown. (c) The final reconstruction results in [101]. (d) The initial reconstruction results in our algorithm. The union of cylinders in each zone is shown. (e) The final reconstruction results in our algorithm.

To get rid of the affection of the geometric agency and produce gradual shape change

during progressive modeling, we used a CSG (Constructive Solid Geometry)-based method, instead of the projection-based method to generate the sub-surface in each zone. Our method is inspired by the work [132], which builds a 3D model from 2D silhouettes sketched under orthogonal views by generating cylinders from each silhouette and computing the union of them. However, the inputs in our system are 3D cross sections with different depths and complex mutual relationship, so the computation will be more involved.

Generally, in each zone zn_i, we build a cylinder from each cross section cs_p and then compute the union of these cylinders. Meanwhile, we extend some cylinders which may probably become isolated surface components, to make them intersect with cylinders in other zones and thus reduce the possibility of the existence of multiple components of the final surface. The final surface is then obtained by stitching all pieces of the sub-surfaces built in each zone.

Specifically, depending on the number of faces having cross section curves on, a zone is first classified into one of the following three types:

- A zone with no faces containing cross sections (denoted by an *empty* zone),
- A zone with only one face containing cross sections (denoted by an *end* zone),
- A zone with two or more faces containing cross sections (denoted by a *body* zone).

Figure 4.1 is a 2D illustration of such a partition with three types of zones. The red segments on the lines correspond to the cross sections on the zone faces.

In the next three sections, we will show how to create sub-surfaces in the *body* zones and *end* zones and how to stitch the sub-surfaces. For an *empty* zone, it has no cross sections and is thus ignored.

4.3 Sub-surface reconstruction for a *body* zone

For a cross section cs_p on face f_{ij} which does not intersect with other cross sections in zone zn_i, we need to build a cylinder cl_p taking the 2D region enclosed by cs_p as the bottom cl_p^b.

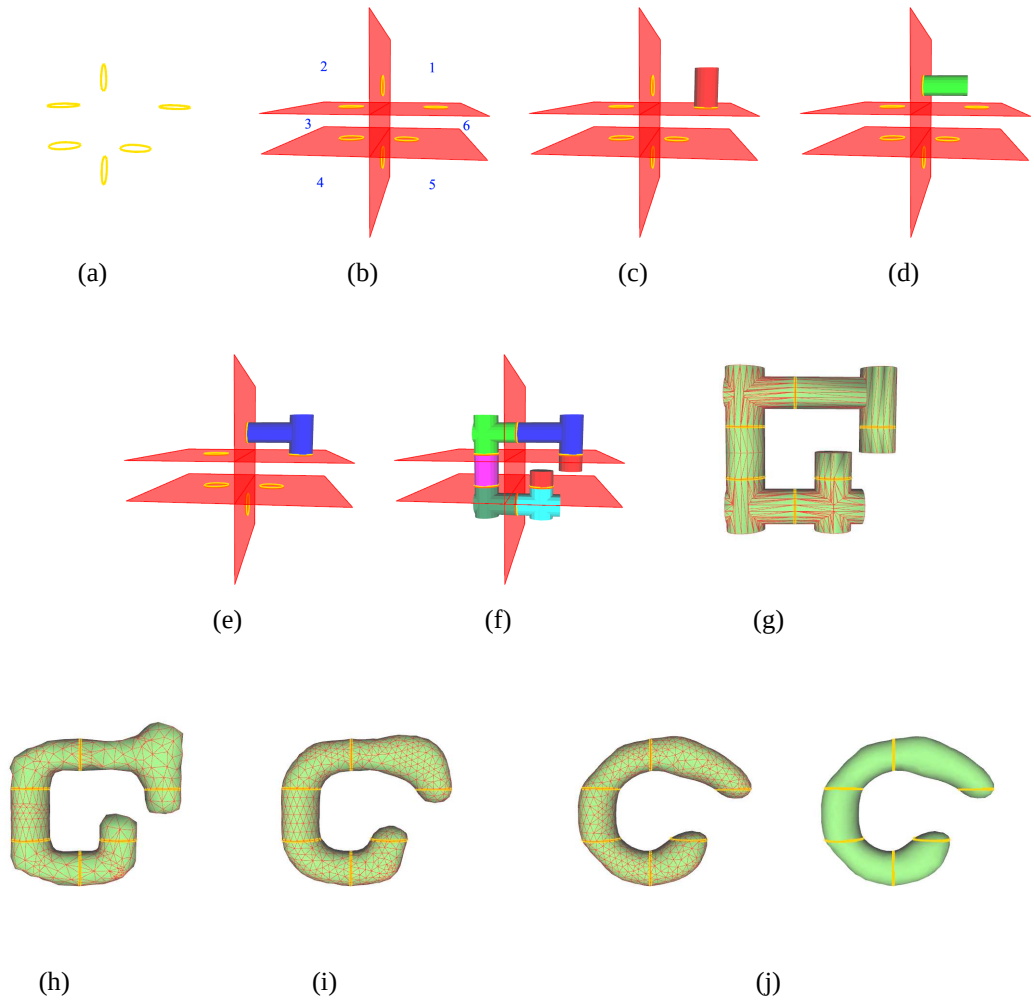


Figure 4.3: An example of surface reconstruction from non-intersected cross sections in the *body* zones. (a) The input cross sections. (b) Partition result. The red planes are the bounding planes of each zone and the numbers represent the indices of the zones. (c)-(d) Cylinders generated from the two cross sections in Zone 1. (e) The union of cylinders in Zone 1. (f) The unions of cylinders in all zones. (g) The initial reconstruction result obtained by stitching surfaces in all zones. (h)-(i) The results after 2 and 5 iterations of refinement and smoothing. (j) The final reconstruction result after 10 iterations of refinement and smoothing.

The initial top cl_p^t of cl_p is obtained by translating cl_p^b along the direction orthogonal to f_{ij} and pointing towards the inside of zn_i. Suppose the height of the zone is h_{zn} regarding f_{ij} as the bottom face of the zone (as we have introduced, each zone is a cuboid), the distance of the translation h_p (the height of cl_p) is then computed as:

$$h_p = h_{zn} - \epsilon, \tag{4.1}$$

where ϵ is a small positive number to make sure that the center of cl_p^t lie within zn_i. In this way, the cylinder cl_p will be totally contained in the zone zn_i and interpolates the input cross section cs_p. An example of the generation of the cylinders in this case can be found in Figure 4.3.

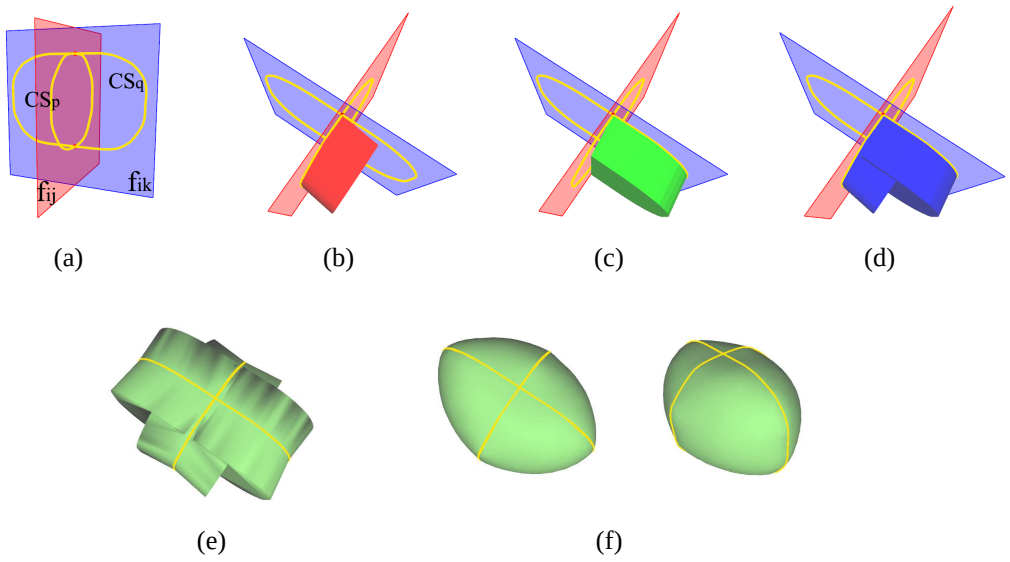
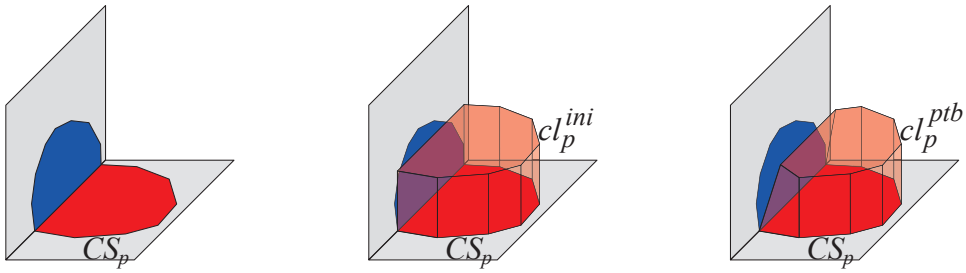


Figure 4.4: An example of surface reconstruction from intersected cross sections in the *body* zones. (a) The input cross sections. (b)-(c) Cylinders are generated from the two cross sections and perturbed. (d) The union of cylinders in the zone. (e) The initial reconstruction result. (f) The final reconstruction result after refinement and smoothing.

When a cross section cs_p on face f_{ij} intersects with another cross section cs_q on the neighbor face f_{ik} of f_{ij} in zone zn_i (see Figure 4.4(a)), computing the cylinder cl_p in the above method will make one face of cl_p coincide with f_{ik}. As a result, the surface of the union of the cylinders may probably not interpolate cs_q on f_{ik}. In that case, we first generate an initial cylinder cl_p^{ini} from cs_p in the above method. Then we perturb cl_p^{ini} a little bit, by

translating a subset of points of cl_p^t which lie on the neighbor face f_{ik} towards the center point of cl_p^t a small distance, such that there will not be any face of the result cylinder cl_p^{ptb} coincide with f_{ik}, and interpolation of the final surface on cs_q can thus be achieved. An illustration of this perturbation process can be found in Figure 4.5. Strictly speaking, cl_p^{ptb} will not be a cylinder anymore, since the shapes of its bottom and top are not the same after the perturbation. However, this will not affect the union calculation which takes general polyhedral as input. Similarly, the cylinder cl_q^{ptb} can be generated from cs_q. The result surface after the union calculation will lie in zn_i and interpolate the input cross sections (see Figure 4.4(d)). This perturbation method also works for the generation of a cylinder from a cross section which intersects with other cross sections on multiple neighbor faces.



(a) (b) (c)

Figure 4.5: Illustration of constructing a cylinder for intersected cross sections in a *body* zone. (a) Input cross sections. (b) Generate initial cylinder cl_p^{ini} from one cross section cs_p. It can be seen that the other cross section is covered by cl_p^{ini}. (c) Perturb the top of cl_p^{ini} a little bit, such that the result cylinder cl_p^{ptb} will lie within the zone and interpolate cs_p.

4.4 Sub-surface reconstruction for an *end* zone

Similar to the method in [101], we use the classical partition method in Computational Geometry to divide the whole space into zones and build sub-surfaces within each zone. However, this strategy may cause the reconstructed model to have some isolated components, because some faces which do not have cross sections on will still take effect on forming the *end* zones (the blue faces in Figure 4.6(c)), and the cylinders generated in these zones cannot connect to any other surface components except those generated in the

other incident zones of the faces containing the cross sections. As a result, these cylinders may probably become isolated surface components and the whole surface will thus be composed of multiple components(see the 2D example in Figure 4.1(d) and 3D example in Figures 4.6(f)).

In progressive modeling, when the user gradually adds new sketches that are closely related to the existing shape, the algorithm is desired to produce a model with only one component. This is a reasonable criterion that agrees with user's expectation. Therefore, we adopt a strategy to extend the cylinders built in the *end* zones to other non-end zones (the *body* and *empty* zones), to minimize their possibility of becoming isolated surface components.

Specifically, for each cross section cs_p on face f_{ij} in an *end* zone zn_i, we first build a cylinder cl_p from cs_p using the same method as generating a cylinder from a non-intersected cross section in a *body* zone. Then we check if the face f_{ik} opposite to f_{ij} in zn_i belongs to the bounding box or not. If yes, then the extension will be considered as impossible and cl_p will be treated as the surface component generated from cs_p in zn_i.

If f_{ik} does not belong to the bounding box, the extension is regarded as possible. In that case, cl_p will be stretched into the neighbor zone zn_k of zn_i that f_{ik} incident to, along the direction orthogonal to f_{ij}. The height of the cl_p will then taken as that of the zone zn_i (regarding f_{ij} as the bottom), plus the computed height in zone zn_k using Eq. 4.1 (regarding f_{ik} as the bottom). If the zone zn_k is a body zone, or the face opposite to f_{ik} in zn_k belongs to the bounding box, this extension process will terminate and the extended cl_p will be regarded as the surface components generated from cs_p; Otherwise, cl_p will be further extended in the same way until these requirements are satisfied.

As shown in Figure 4.6, by extending the cylinders built in the *end* zones to make them intersect with other cylinders, isolated surface components can be connected to form a single component. While for the projection-based method in [101], this extension is not easy to implement, and the result surface will contain multiple isolated components.

It should be pointed out that this cylinder extension strategy cannot guarantee the final

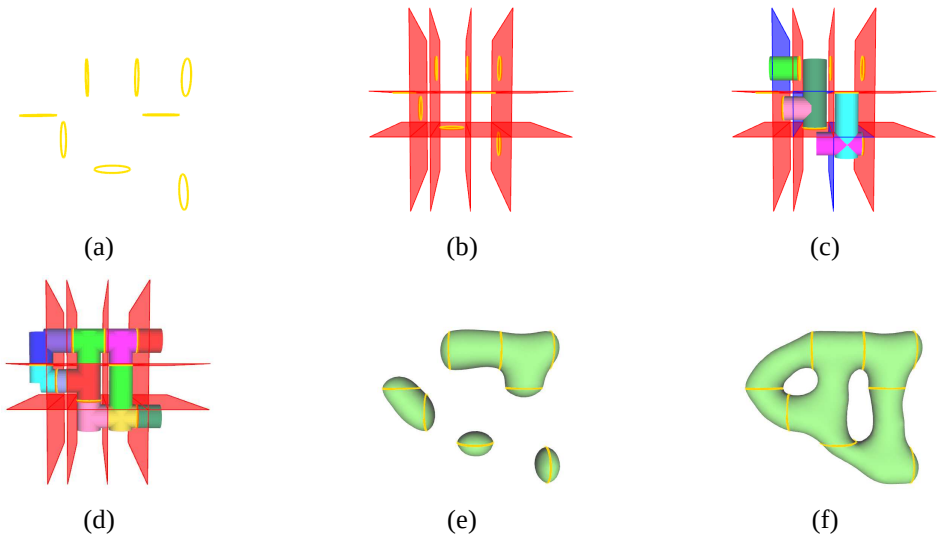


Figure 4.6: An example of surface reconstruction in the *end* and *body* zones. (a) Input cross sections. (b) The partition result. (c) The extended cylinders for the *end* zones. The blue faces are those which may cause isolated surface components. (d) The unions of cylinders in all zones. (e) Final surface reconstruction result in our approach. (f) Reconstruction result using the algorithm in [101].

surface to always have a single component. Such cases may happen when cylinders within the same zone cannot intersect whatever their heights are (such as the two cylinders in Zone 6 in Figure 4.3(f)). Although this problem can be fixed by changing the shape of the cylinders to force them connect regardless of the positions of the sketched cross sections, we do not launch such a process since it may produce irregular and undesired local shapes. Meanwhile, since the user usually tends to add a new sketch that is closely related to the current shape, if an isolated part is to be generated during the sketching process, a hint will be given to the user to indicate that possibility caused by the newly added sketch.

4.5 Global surface reconstruction

After all the cylinders in each zone are built, we use the CarveCSG library [4] which implements the algorithm proposed in [91] to compute the union of these polyhedral one by one.

Given two polyhedral P_a and P_b, each triangle Tr_i^a on P_a will be visited first. If Tr_i^a

intersects with a triangle Tr_j^b on P_b, then both Tr_i^a and Tr_j^b will be bisected by the plane of each other's. This way, P_a and P_b will become new polyhedral P_a' and P_b' which are composed of bisected triangles. Next, the triangles on P_a' which are inside the volume of P_b', as well as those on P_b' inside the volume of P_a' will be removed. Finally, the left triangles from P_a' and P_b' will be merged and the union result can be obtained.

This method is robust and quick enough for our application since the shape of the polyhedral are regular. The global surface of the initial mesh M_{init} can then be calculated as:

$$M_{init} = \sum_{i=1}^{m} (\bigcup_{j=1}^{n} Cl_{ij}) \tag{4.2}$$

where Cl_{ij} refers to the jth cylinder built in the ith zone, assuming there are totally m zones and n cylinders in each zone.

4.5.1 Mesh improvement

Since the shape of the initially reconstructed surface M_{init} is usually unnatural and jaggy, we then improve it through iterative refinement and smoothing.

For the mesh refinement, we used the algorithm proposed in [96] to produce a result similar to the Delaunay-like triangulation through iterative triangle splitting and edge swapping.

First, we compute a length attribute σ_i for each vertex v_i^0 on the input cross sections as the average length of all the edges adjacent to v_i^0. Then the attribute for each neighbor vertex v_i^1 which is not on the input cross sections will be computed as the average of σ_i of all v_i^0s which are neighbors of v_i^1. The length attributes for all other vertices will be computed ring by ring in this propagation way.

Then, for each triangle (v_i, v_j, v_k), we compute its centroid v_c and the length attribute σ_c for v_c as $\sigma_c = (\sigma_i + \sigma_j + \sigma_k)/3$. For any vertex v_m ($m \in \{i, j, k\}$) of this triangle, if $\alpha||v_m - v_c|| > \sigma_c$ and $\alpha||v_m - v_c|| > \sigma_m$ (we set $\alpha = \sqrt{2}$ as suggested in [96]), we split the triangle (v_i, v_j, v_k) to triangles (v_i, v_j, v_c), (v_j, v_k, v_c) and (v_k, v_i, v_c), and relax the edges

(v_i, v_j), (v_j, v_k) and (v_k, v_i).

Here relaxing an edge (v_i, v_j) means for the two triangles (v_i, v_j, v_k) and (v_i, v_j, v_n) adjacent to (v_i, v_j), we check if v_k and v_n lie inside the circumcircles of triangles (v_i, v_j, v_n) and (v_i, v_j, v_k) respectively. If yes, we implement the edge swapping to let the two triangles become (v_i, v_k, v_n) and (v_j, v_k, v_n). To ensure the interpolation on the input curves, the edge swapping is restricted to the edges which do not belong to the input cross sections.

If there are new triangles created in the triangle splitting step, we relax all the edges on the mesh surface and repeat the splitting step; otherwise, the refinement process is complete.

For the mesh smoothing, we adopted the algorithm in [165] to produce meshes with smooth shapes. The position p_i of each vertex v_i is updated to p_i' through the following equation:

$$p_i' = p_i + \tau_i N_i + \epsilon_i T_i. \tag{4.3}$$

In the above equation, N_i is the unit normal vector at v_i. We compute N_i as the normalized weighted sum of the normals of the incident triangles of v_i, with weights equal to the areas of the triangles. We set the factor $\tau_i = A_i^2/150$ as suggested in [165], where A_i is the minimum area among all the incident triangles of v_i. To improve the stability of the surface evolution, we then move the vertex along a tangential vector T_i to make the triangle sizes as equal as possible. More precisely, we compute it as $T_i = \delta_i + N_i$, where δ_i is the vector pointing from v_i to the centroid of all its neighbor vertices. The factor ϵ_i is set to 0.5 in Eq. 4.3. Similarly, the position update is only limited to the vertices which do not lie on the input cross sections to ensure the interpolation.

In our implementation, we alternatively refine and smooth the initial mesh for 10 iterations. The result surface will have satisfactory quality and its interpolation on the input cross sections is maintained. An example of the refinement and smoothing process can be found in Figure 4.3.

4.6 Results and discussions

Examples produced by our sketch-based modeling system using the CSG-based progressive surface reconstruction algorithm can be found in Figure 3.7.

By making use of the cylinders which reflect the shape and positions of the sketched cross sections in the reconstruction process, our approach can get rid of the MA plane, whose change is more complicated during progressive sketching and will lead to unexpected shape change of the surface parts generated from unrelated cross sections. As a result, the change of the model shape will be natural under any sketching orders, which meets Requirement 1 for our sketching system. An example showing this advantage of our CSG-based method over the projection-based method in [101] can be found in Figure 4.2.

In the reconstruction process, the extension of the cylinders in the isolated zones, as well as the union calculation of the cylinders in all zones are carried out independently. So for a same set of sketches, the final reconstruction result will be unique and insensitive to its input orders.

There are also some other advantages of the CSG-based method which have been utilized in our progressive modeling interface. For example, since the calculations within each zone are independent, each time the user adds, modifies or deletes a cross section, only the sub-surfaces in the affected zones need to be recalculated and updated, while those in other zones can be retained. In this way, the calculation can be implemented locally and accelerated.

The independence of calculation in each zone also makes it possible to allow the sketching of only a part of the cross section instead of a complete one, as long as the stroke rules are satisfied within the zone. Since the user sketches are more or less arbitrary, we also perform automatic curve sewing similar to that mentioned in Section 3.3.2 or give user related hints, to make sure the stroke rules are followed.

Although the CSG-based progressive surface reconstruction algorithm is designed for the sketching operation in our system, it can also be used in other applications such as

the CT or ultrasound scan, in which the input cross sections are obtained from gradually scanned slices. By making use of our algorithm, the reconstructed 3D model can be built progressively and its shape can be updated in a natural manner to provide the user desired real-time feedback.

Chapter 5

Triangular Mesh Deformation via Edge-Based Graph

This chapter presents an edge-based flexible mesh deformation algorithm, which effectively supports the deformation function in our sketch-based modeling system.

5.1 Introduction

Mesh deformation is an effective and convenient method for mesh editing in geometric modeling and computer animation, and it is also an important sculpting function in sketch-based modeling systems. It allows the user to manipulate one part or the entire surface while preserving some geometric characteristics of the initial surface. As introduced in Section 2.2, various algorithms on mesh deformation have been developed due to its importance and popularity in a wide rage of applications in industrial and artistic designs. Though having their respective advantages, those algorithms also have drawbacks due to the following considerations in mesh deformation.

First, a triangular mesh is a piecewise linear surface and it is often considered as an approximation of some unknown smooth surface. The approximation effect is related to the number of vertices of the mesh. In general, the larger the number of vertices the mesh

contains, the better is the approximation. When a deformation is performed on a surface, the calculation will be conducted on the vertices. The number of vertices determines the sampling rate in approximating a continuous model and thus a denser mesh usually results in more accurate deformation. However, given a deformation region, the number of vertices is fixed and it is not trivial to have more points be involved in the deformation calculation.

Second, in the deformation process the local shape feature is often required to be well preserved and the deformed shape should be natural and smooth to satisfy aesthetics or other requirements. A natural problem then arises: what is a good balance between the rigidity and fairness? The answer is subjective. A satisfactory editing result depends on the algorithm adopted, the specific models and the desired effect as well.

Third, an object in real world may be composed of different materials, which may exhibit different behaviors when the object is deformed. For example, with different stiffness properties of the materials, different deformation effects between rigidity and smoothness will appear under the deformation. Most of the existing deformation methods are purely geometrically-based and they ignore the material properties of the object.

To solve these problems and make the deformation more flexible, we propose a flexible mesh deformation algorithm which performs the computation through an edge-based graph. Instead of formulating the deformation on the primal domain, we build an edge-based graph and carry out the deformation computations on the graph, which involves more nodes and thus produces better deformation results. We then propose a mixed energy model that combines the first order and the second order discrete differential quantities to balance local shape preservation and smoothness. The user can adjust the values of the balance parameter to control the deformation results. Moreover, these introduced balance parameters are used to reflect the stiffness property of the object material and thus make the deformation algorithm material-aware. A simple sketching tool is provided to specify the stiffness property. The novelty of the chapter thus lies in the use of the edge-based graph for the deformation formulation and the use of balance parameters for controlling the stiffness and producing various reasonable deformation results in a mesh deformation task.

5.2 Edge-based graph

Let $M = (V, E)$ be a triangular mesh with n vertices. $V = \{v_i | v_i \in R^3, i = 1, ..., n\}$ denotes the set of vertices and $E = \{e_{ij} = (v_i, v_j) | v_i, v_j \in V, i \neq j\}$ represents the set of edges. Correspondingly, we define $\tilde{M} = (\tilde{V}, \tilde{E})$ as the corresponding edge-based graph for M, where $\tilde{V} = \{\tilde{v_i} | \tilde{v_i} \in R^3, i = 1, ..., n_e\}$ and $\tilde{E} = \{\tilde{e_{ij}} = (\tilde{v_i}, \tilde{v_j}) | \tilde{v_i}, \tilde{v_j} \in \tilde{V}, i \neq j\}$ are the sets of nodes and edges of the edge-based graph respectively. Specifically, a node $\tilde{v_i}$ is defined as the midpoint of the edge e_{ij} in the primal domain: $\tilde{v_i} = (v_i + v_j)/2$. n_e represents the number of nodes in the edge-based graph, which equals the number of edges on the primal mesh M. Each node $\tilde{v_i}$ corresponding to primal edge e_{ij} is connected to four neighboring nodes corresponding to the four primal edges that share the vertices of e_{ij}. In this way, the valence of each node in the edge-based graph is always four. The conversion from the primal vertex positions V to the edge-based graph vertex positions $\tilde{V}$ can be accomplished by an $n_e \times n$ sparse matrix D. Each row of D has two non-zero elements that are 1/2. Figure 5.1 shows an edge-based graph constructed from a primal mesh.

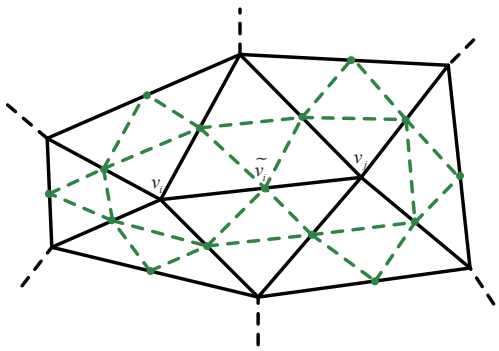


Figure 5.1: An illustration of an edge-based graph. The black lines represent the primal mesh and the green lines form the edge-based graph.

5.3 Flexible mesh deformation

In mesh deformation, the user first specifies the *handle* vertices and their new positions, as well as the *static* vertices that will remain unchanged, and then the algorithm determines the deformation of the *region-of-interest (ROI)* vertices. Many previous deformation algorithms aim to preserve either the first or the second order discrete differential quantities of the mesh surface and thus can produce only one kind of deformation result, which may be either too rigid or smooth.

We present a method that considers both of the two differential quantities and balances between the rigidity and smoothness of the deformation in a flexible manner. Our basic idea is to deform the vertices in the ROI such that a mixed energy functional is minimized and this minimization problem is formulated on an edge-based graph. The mixed energy functional includes the change of the first order and second order differential quantities that reflect stretching and bending. Based on Euler-Poincare formula, the number of edges on a triangular mesh is approximately three times that of vertices. Thus the energy functional formulated on the edge-based graph better approximates the continuous version of the energy functional of the smooth surface than that on the primal domain. Thus the formulation on the edge-based graph tends to render better deformation results, especially when rigid or smooth deformation effects are required. See Figure 5.2 for an example.

5.3.1 First and second order discrete differential quantities

First of all we need to find proper representations for the first and second order discrete differential quantities of the mesh surface.

For a smooth surface, the first order differential properties are usually represented by length, area, etc. However, these will make the system nonlinear. So for triangular meshes we choose to use edge vectors at each node, which we called the 1-form, to represent the first order differential structures of the mesh surface. The 1-form contains the information of edge lengths and can help to make the final system a linear one.

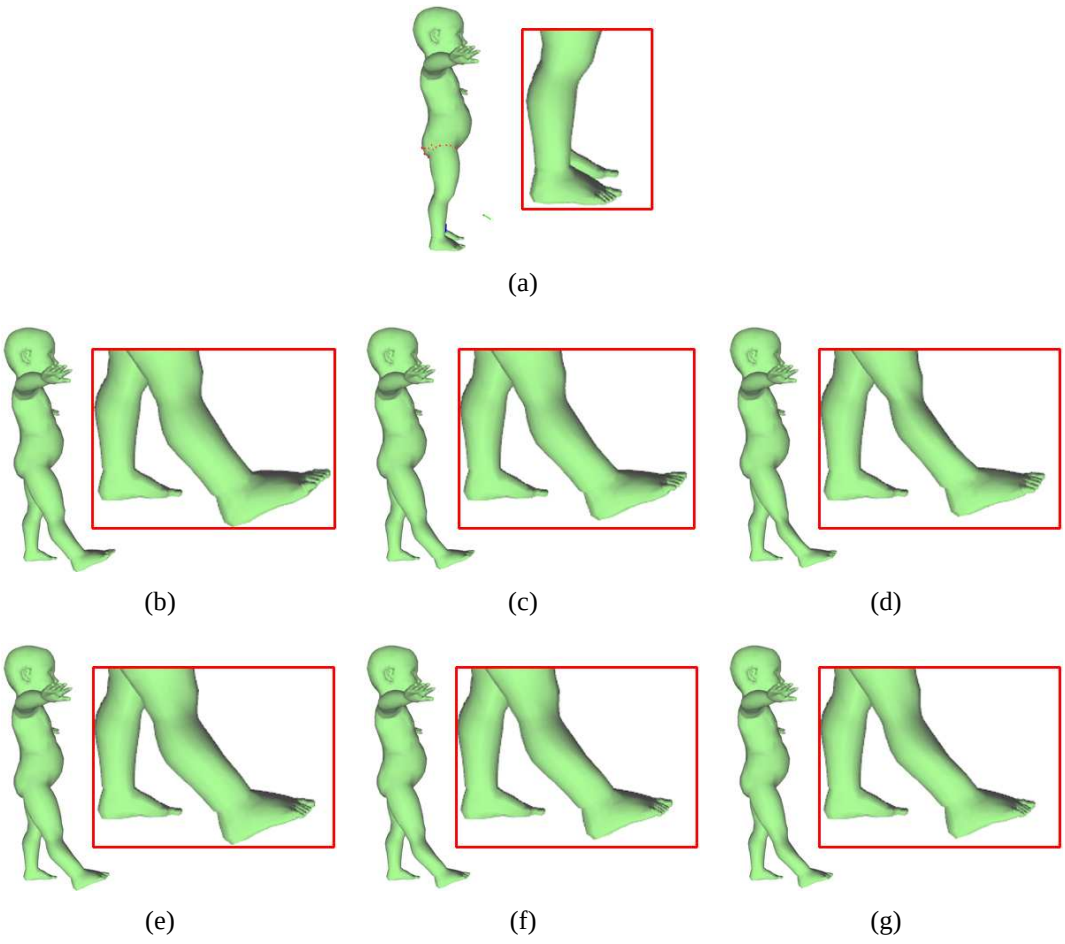
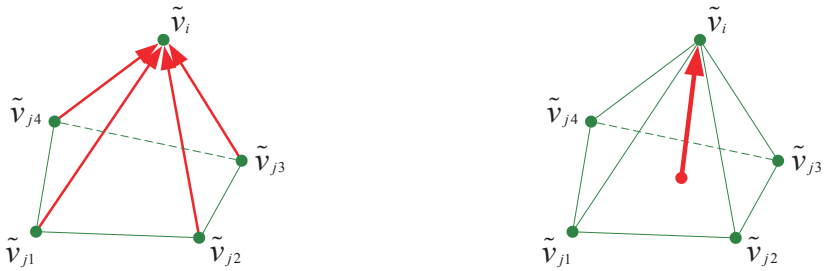


Figure 5.2: Flexible deformation of the *Baby* model. (a) The initial model. (b)-(d) Results of the flexible deformation algorithm performed on the primal domain, with the global balance parameters valued 0, 0.5 and 1.0 respectively. (e)-(g) Results of the flexible deformation algorithm performed via the edge-based graph, with the global balance parameters valued 0, 0.5 and 1.0 respectively.

Specifically, for a node $\tilde{v}_i$, the 1-form can be represented by the edge vectors $\tilde{e_{ij}}$ which start from the one-ring neighborhood $\tilde{v}_j, \tilde{e_{ij}} \in \tilde{E}$ and point to $\tilde{v}_i$, as shown in Figure 5.3(a). It can totally decide the shape of the local area around a node. That is to say, as long as the edge vectors connecting $\tilde{v}_i$ and its neighbors are fixed, the shape of the local area around $\tilde{v}_i$ is fixed.

The second order differential properties of a smooth surface are usually represented by curvatures, which also make the system a nonlinear one [49]. The Laplacian vector provides a proper choice for representing the second order differential properties of the mesh surface, since its magnitude approximates the mean curvature at each vertex and the final system formed is a linear one. So we extend the traditional definition of Laplacian vector from the primal mesh to the edge-based graph and use it to represent the second order discrete differential quantity of the mesh surface, as shown in Figure 5.3(b).



(a) (b)

Figure 5.3: (a) The 1-form. (b) The Laplacian vector.

Mathematically, the Laplacian vector $\tilde{\delta}_i$ at $\tilde{v}_i$ is defined as:

$$\tilde{\delta}_i = L(\tilde{v}_i) = \sum_{j \in N(i)} \tilde{\omega_{ij}}(\tilde{v}_i - \tilde{v}_j), \tag{5.1}$$

where $N(i) = \{j | \tilde{e_{ij}} \in \tilde{E}\}$ is the index set of adjacent nodes in the neighboring ring of $\tilde{v}_i$ and $\tilde{\omega_{ij}}$ is the weight of the edge $\tilde{e_{ij}}$. Here we use the cotangent weight scheme in [108] to compute $\tilde{\omega_{ij}}$:

$$\tilde{\omega_{ij}} = \frac{\cot \tilde{\alpha_{ij}} + \cot \tilde{\beta_{ij}}}{2\tilde{A}_i}, \tag{5.2}$$

where $\tilde{\alpha_{ij}}$ and $\tilde{\beta_{ij}}$ are the two angles opposite to the edge $\tilde{e_{ij}}$. $\tilde{A_i}$ is the Voronoi area of the node $\tilde{v_i}$, which is computed as the area of the surface region built by connecting incident edges' midpoints with triangle circumcenters (for acute triangles) or midpoints of opposite edges (for obtuse triangles) [108], as shown in Figure 5.4. In fact, $L(\cdot)$ can be regarded as the discretized Laplace-Beltrami operator with cotangent coefficients on triangular meshes [28].

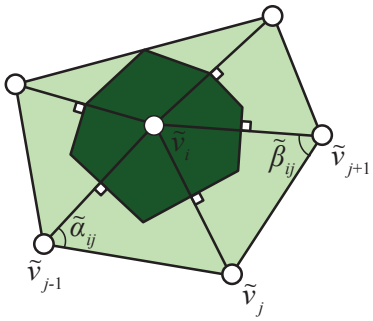
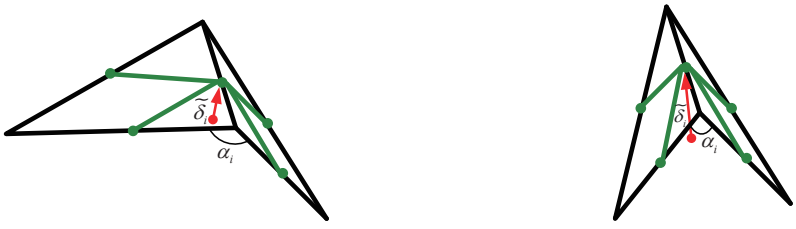


Figure 5.4: The angles $\tilde{\alpha_{ij}}$ and $\tilde{\beta_{ij}}$, and Voronoi area $\tilde{A_i}$ for computing the cotangent weight at the node $\tilde{v_i}$ in Eq. 5.2.

It should be noted that the Laplacian vector $\tilde{\delta_i}$ in the edge-based graph can also be regarded as a measurement of the dihedral angle α_i of two adjacent triangles in the primal mesh, which also represents the second order differential property of the mesh surface. The magnitude of $\tilde{\delta_i}$ is inversely proportional to the size of α_i, as can be seen in Figure 5.5.



(a) (b)

Figure 5.5: The Laplacian vector $\tilde{\delta_i}$ in the edge-based graph and the dihedral angle α_i between adjacent faces in the primal domain. (a) The shorter $\tilde{\delta_i}$ is, the larger α_i will be. (b) The longer $\tilde{\delta_i}$ is, the smaller α_i will be.

These two representations of discrete differential quantities encode the relative position of each node w.r.t. its neighboring nodes and represent the local geometry properties of the

mesh surface. Next we will introduce how to combine them to get a satisfactory deformation result flexibly.

5.3.2 Energy function for flexible deformation

The approach for solving the deformation problem of the surface-based method is to designate the positional constraints for the handle and static vertices and solve for the positions of the ROI vertices by fitting the local geometry properties of the deformed mesh to those of the initial mesh. The properties we want to preserve include both the first and second order discrete differential properties, which are represented by the 1-form and Laplacian vectors.

Since the 1-form at each node $\tilde{v_i}$ can totally decide the shape of a local neighborhood area, it is considered as defining the local shape of the mesh surface in a unique manner. Mathematically, to preserve the 1-form of the initial mesh, the energy we try to minimize for each node $\tilde{v_i}$ is:

$$E(\tilde{v_i}') = \sum_{j \in N(i)} \tilde{\omega_{ij}} ||\tilde{e_{ij}}' - R_i \tilde{e_{ij}}||^2, \tag{5.3}$$

where $\tilde{v_i}'$ is the new position of $\tilde{v_i}$, $\tilde{e_{ij}} = \tilde{v_i} - \tilde{v_j}$ and $\tilde{e_{ij}}' = \tilde{v_i}' - \tilde{v_j}'$ are the undeformed and deformed edge vectors between $\tilde{v_i}$ and $\tilde{v_i}$ respectively. $\tilde{\omega_{ij}}$ is the same weight as that in Eq. 5.1. R_i represents the rotation matrix associated to $\tilde{v_i}$.

Given the old and new values of the edge vectors, we use the method proposed in [144] to compute the optimal rotation matrix R_i which minimizes Eq. 5.3. First, Eq. 5.3 can be re-written as:

$$
\begin{aligned}
E(\tilde{v_i}') &= \sum_{j \in N(i)} \tilde{\omega_{ij}} ||\tilde{e_{ij}}' - R_i \tilde{e_{ij}}||^2 \\
&= \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{e_{ij}}' - R_i \tilde{e_{ij}})^T (\tilde{e_{ij}}' - R_i \tilde{e_{ij}}) \\
&= \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{e_{ij}}'^T \tilde{e_{ij}}' - 2\tilde{e_{ij}}'^T R_i \tilde{e_{ij}} + \tilde{e_{ij}}^T R_i^T R_i \tilde{e_{ij}}) \\
&= \sum_{j \in N(i)} (\tilde{\omega_{ij}} \tilde{e_{ij}}'^T \tilde{e_{ij}}' - 2\tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}} + \tilde{\omega_{ij}} \tilde{e_{ij}}^T \tilde{e_{ij}}) \qquad (5.4)
\end{aligned}
$$

Since the terms that do not contain R_i are constant, they can be ignored. The goal then is to compute the optimal R_i which minimizes the term $\sum_{j \in N(i)} -2\tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}}$ (i.e. maximizes $\sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}}$). The matrix $\sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}}$ is a diagonal matrix, and the sum of the elements on the diagonal equals to its trace when the matrix is a square one, so the problem turns to solving for the optimal matrix R_i that maximizes $tr(\sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}})$, where $tr(\cdot)$ represents the trace of the matrix.

According to the property of the trace of the matrix, $tr(\sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T R_i \tilde{e_{ij}})$ equals to $tr(R_i \sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T \tilde{e_{ij}})$. For simplicity, we denote the matrix $\sum_{j \in N(i)} \tilde{\omega_{ij}} \tilde{e_{ij}}'^T \tilde{e_{ij}}$ by C_i. It is well known that the rotation matrix R_i maximizing $tr(R_i C_i)$ is obtained when $R_i C_i$ is symmetric positive semi-definite (if M is a positive semi-definite matrix then for any orthogonal R, $tr(M) \geq tr(RM)$). Thus, the optimal rotation matrix R_i can be derived from the singular value decomposition of $C_i = U_i \Sigma_i V_i^T$:

$$
R_i = V_i U_i^T, \qquad (5.5)
$$

up to changing the sign of the column of U_i corresponding to the smallest singular value, such that the determinant of R_i will be positive.

The rigid transformation of the 1-form during deformation strives to do a *hard* transform to the mesh. We say *hard* as the edge vectors of each node can completely determine the shape of its neighborhood area. As long as each vector keeps fixed, the shape of this area

will not change at all.

To preserve each Laplacian vector of the mesh surface during deformation, we try to minimize its change up to an affine transformation T_i. Since shearing and anisotropic scale will lead to distortions of the local features [54], we assume T_i be composed of rotation and isotropic scale, i.e. $T_i = s_i R_i$. s_i and R_i represent the scale coefficient and rotation matrix separately. The energy to minimize is:

$$E(\tilde{v_i}') = || \sum_{j \in N(i)} \tilde{\omega_{ij}}(\tilde{v_i}' - \tilde{v_j}') - T_i \sum_{j \in N(i)} \tilde{\omega_{ij}}(\tilde{v_i} - \tilde{v_j})||^2. \tag{5.6}$$

The deformation of preserving the Laplacian vectors is a relatively *soft* deformation, comparing to the *hard* one, since the Laplacian vector cannot exactly represent the shape of the local area around each node. The word *soft* has another meaning that the deformed shape tends to be smooth, since the change of the second order differential quantity is minimized to prevent the sharp change of the mesh shape within a local area.

Up to now, we have obtained two linear methods for mesh deformation by preserving the 1-form and Laplacian vectors respectively. The unknowns in the two systems are both the set of deformed node positions. However, neither of the two methods can satisfy our requirement of obtaining various deformation effects between rigidity and smoothness. In the first method, while the representation of the local shape is quite accurate, sometimes the deformation will be too rigid that smoothness cannot be obtained. Besides, a definite rigid transformation means that scale is not considered, and thus some local features cannot be preserved under this kind of transformation. In the second method, since the Laplacian vector cannot exactly represent the shape of a local region, local shape preservation may not be achieved. However, the deformation calculated using this method is relatively smooth. Moreover, scale transformation is considered in the computation to make the result more natural and help to preserve the local features under stretching or shrinking.

To make use of the advantages of these two methods and give the user flexibilities to choose the most desirable deformation effect between rigidity and smoothness subject to

a specific model, we propose a new algorithm which combines the two methods using a balance parameter λ. When $\lambda = 1$, only the first order discrete differential quantity of each node is preserved and the deformation is the most rigid; when $\lambda = 0$, only the second order discrete differential quantities are preserved and a smooth result is achieved. Multiple intermediate results can be obtained by setting different λ values between 0 and 1.

Considering the positional constraints $\{\tilde{c_i}\}, i \in \{m, ..., n\}, m < n$, the overall energy we aim to minimize for a mesh deformation task with n nodes involved is:

$$\begin{aligned} E(V') &= \sum_{i=1}^{n} \lambda_i (\sum_{j \in N(i)} \tilde{\omega_{ij}} ||\tilde{e_{ij}}' - R_i \tilde{e_{ij}}||^2) + \\ &\sum_{i=1}^{n} (1-\lambda_i) || \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{v_i}' - \tilde{v_j}') - T_i \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{v_i} - \tilde{v_j}) ||^2 + \\ &\sum_{i=m}^{n} ||\tilde{v_i}' - \tilde{c_i}||^2, \end{aligned} \tag{5.7}$$

in which the first and second items of the right hand side are the energies measured by the change of the 1-form and Laplacian vectors and the third item is the positional constraints during deformation. The balance parameter $\lambda_i (0 \leq \lambda_i \leq 1)$ is used to control the rigidity and smoothness of the deformation at node $\tilde{v_i}$.

It is not difficult to see that the definitions of 1-form and Laplacian vectors and the flexible deformation algorithm can be easily transferred to the primal domain, by replacing the variables defined in the edge-based graph in Eq. 5.7 with those of the primal domain. This is useful for comparing the deformation results calculated directly in the primal domain and via the edge-based graph. Detailed comparisons can be found in Section 5.5.

5.3.3 Solving of the system

To solve this system, we use an iterative method which can be outlined as follows.

First we build an edge-based graph for the deformation region of the mesh and then

compute the partial derivatives of the first and second items in energy function 5.7 w.r.t. the unknown node positions $\tilde{V}'$ and get two linear systems $A_1\tilde{V}' = b_1$ and $A_2\tilde{V}' = b_2$. An important observation is that if the coefficients λ_i and $1 - \lambda_i$ are not considered, A_1 and A_2 happen to be the same. So only one of them needs to be constructed in the beginning and we just need to focus on the calculation of the right hand side of the system in each iteration.

Then an initial deformation result is estimated using the naive Laplacian deformation [9] which does not consider rotation and scale transformations of the Laplacian vectors. Based on the estimated vertex positions, we can compute the rotation matrix R_i and the transformation matrix T_i. Since the optimal R_i can be uniquely calculated through singular value decomposition [144], we make use of it to compute the optimal scale coefficient s_i, such that $T_i = s_i R_i$ can be obtained. This way, computations of R_i and T_i are associated and the overall calculation process is accelerated. The right hand side of the system is then updated by multiplying the calculated R_i and T_i and the system can be solved.

Next some iterations are performed for recomputing R_i and T_i, updating the right hand side of the system and solving it. This process continues until convergence.

As all the above computations are performed in the edge-based graph, we need to reconstruct the primal mesh by using the conversion matrix D introduced in Section 5.2 as well as the positional constraints for primal handle and static vertices. However, since the position of each node in the edge-based graph can be represented as the linear combination of two primal vertex positions, an easier way to solve the system is to integrate D into Eq. 5.7 and directly take the primal vertex positions as unknowns.

5.4 Specification of the balance parameters

In the preceding section we assume that the balance parameters have been predefined and treat them as constants during the calculation. Now we provide two methods to specify the balance parameters: a global method and a local method.

In the global method, we let the balance parameter λ_i for each node $\tilde{v_i}$ equal to the same value λ, such that energy function 5.7 turns to:

$$\begin{aligned} E(V') &= \lambda \sum_{i=1}^{n} (\sum_{j \in N(i)} \tilde{\omega_{ij}} ||\tilde{e_{ij}}' - R_i \tilde{e_{ij}}||^2) + \\ & (1-\lambda) \sum_{i=1}^{n} || \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{v_i}' - \tilde{v_j}') - T_i \sum_{j \in N(i)} \tilde{\omega_{ij}} (\tilde{v_i} - \tilde{v_j}) ||^2 + \\ & \sum_{i=m}^{n} ||\tilde{v_i}' - \tilde{c_i}||^2. \end{aligned} \tag{5.8}$$

This global method provides a proper choice when the user wants to make the entire editing region of a mesh deform in a consistent manner. The rigidity and smoothness of the deformation can be well balanced through setting a proper λ value between 0 and 1. The user can select a proper λ value to produce a satisfactory result, according to the specific model and the deformation effect he needs. Generally speaking, larger λ values make the deformation more rigid and smaller λ values make it more fair. It can be seen that previous methods which preserve either the first or the second order differential properties of the mesh surface is just a special case of Eq. 5.8.

Examples of the global method can be seen in Figures 5.6 to 5.8.

Although the global method provides an efficient way of calculating a natural and consistent deformation result for the entire editing region, it ignores the individual property of each local area on the mesh surface and sometimes is not flexible. So we also provide a local method for the specification of the balance parameters.

Since objects in the real world are often composed of non-uniformly distributed materials, their deformation behaviors vary from rigidity to smoothness across the surface, depending on the stiffness of the materials on each local region. However, current mesh deformation algorithms either ignore the underlying materials of the mesh surface or suffer from various problems. The physics or skeleton based algorithms provide feasible ap-

proaches to solve this problem. Nevertheless, they are either slow in computation or limited to a subset of models [129].

Since the balance parameter can be used to control the rigidity and smoothness of the deformation, we can use it to define the stiffness property of the material on each local area of the mesh surface, such that the regions with different materials will have distinguished behaviors in a same deformation task. In this case, λ_i for each $\tilde{v_i}$ in Eq. 5.7 should be defined separately according to the material property of $\tilde{v_i}$. The larger λ_i is, the stiffer the material will be and the deformation at $\tilde{v_i}$ should be relatively rigid; otherwise, the material will be softer and the deformation should be smooth.

Here we provide sketching tools, which are intuitive and flexible for interactive designs, for the user to designate materials on different parts of the mesh surface. The user can first set a scalar value which represents the stiffness property of the material and then use lasso or painting brush tools to specify the material for the selected part. Since the entire user operations are performed on the surface of the primal mesh while the calculations are implicitly implemented on the edge-based graph, after the materials of the primal vertices are specified, we calculate the material for each node $\tilde{v_i}$ of the edge-based graph by averaging those of the two end vertices of the primal edge e_{ij} that $\tilde{v_i}$ lies on.

Results of the local method can be seen in Figures 5.9 to 5.11.

It can be summarized that when a globally consistent deformation is desired, the global method can be used to produce a well-balanced result between rigidity and smoothness, subject to the specific model and user requirement; when the user wants to compute the deformation of a mesh surface with non-uniformly distributed materials, then the local method can be used to specify the stiffness properties of the materials on the local regions and get a natural and realistic result.

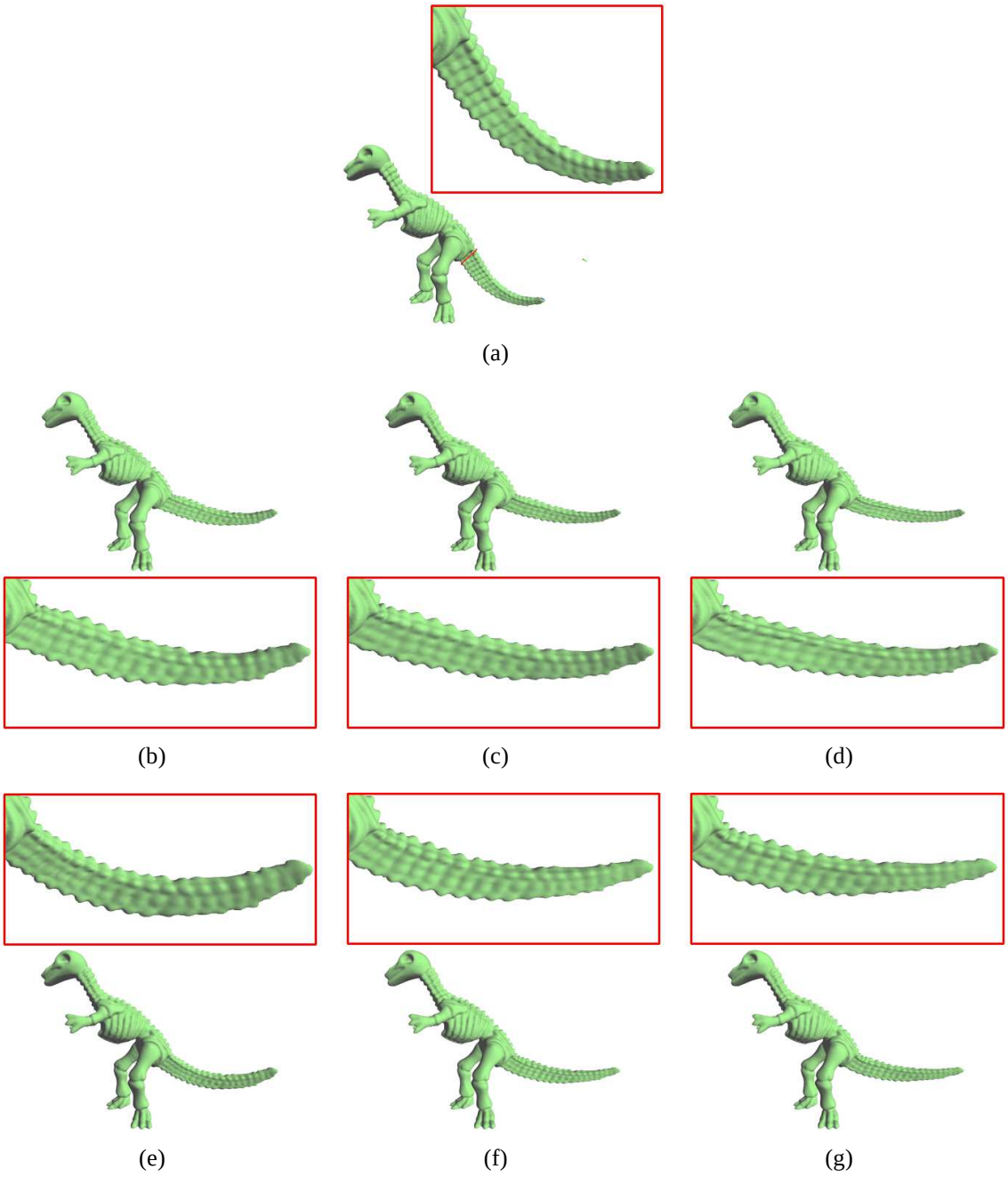


Figure 5.6: Flexible deformation of the *Dinosaur* model. (a) The initial model. (b)-(d) Results of flexible deformation algorithm performed on the primal domain, with the global balance parameters valued 0, 0.5 and 1.0 respectively. (e)-(g) Results of flexible deformation algorithm performed via the edge-based graph, with the global balance parameters valued 0, 0.5 and 1.0 respectively.

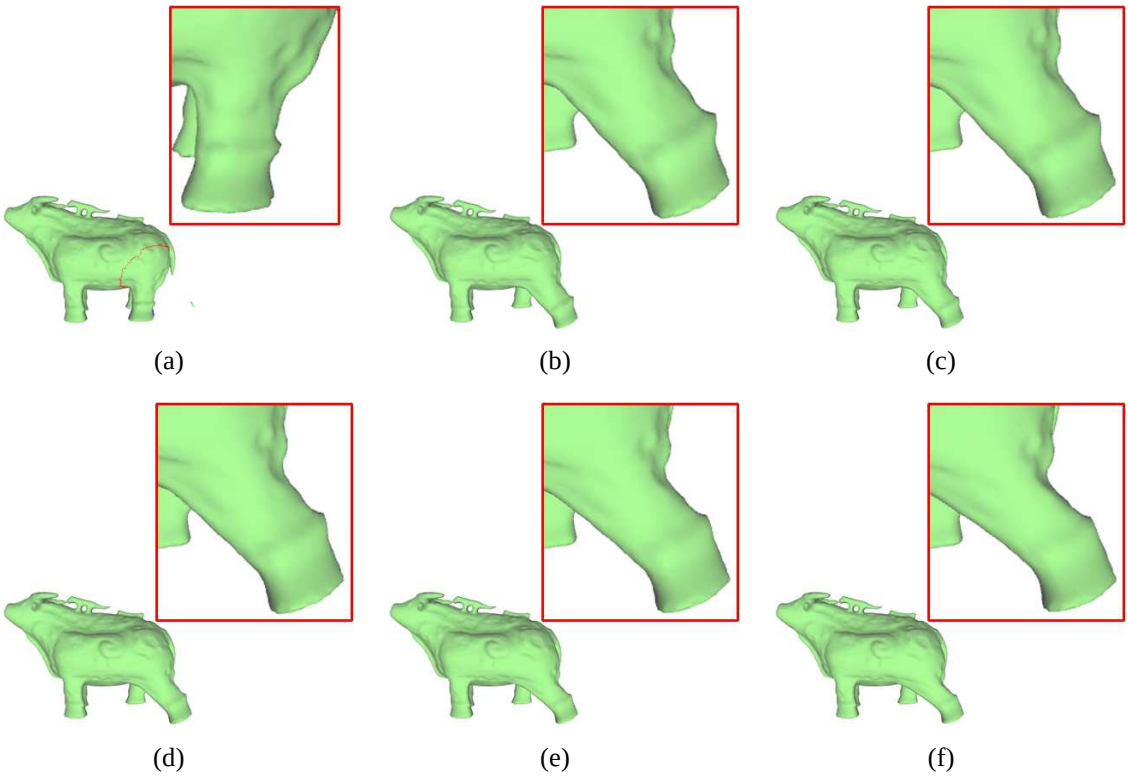
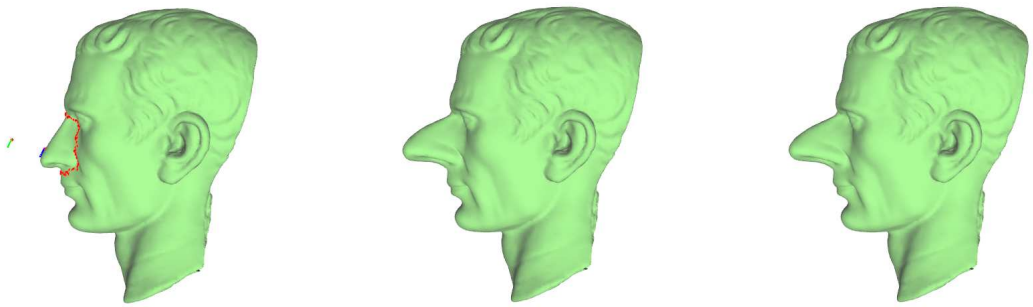
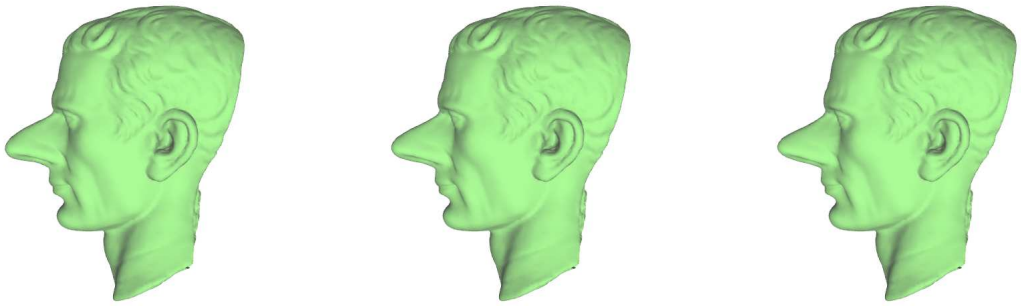


Figure 5.7: Flexible deformation of the *Buffle* model via the edge-based graph under different balance parameter values. (a) The initial model. (b)-(f) Deformation results with the global balance parameters valued 0, 0.2, 0.5, 0.8, 1.0 respectively.

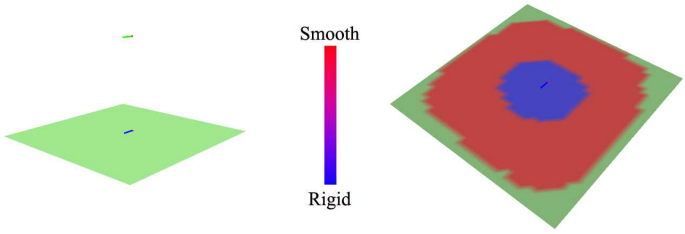


(a) (b) (c)

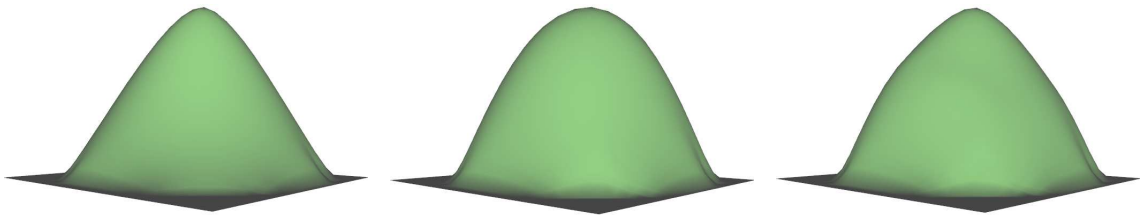


(d) (e) (f)

Figure 5.8: Flexible deformation of the *Julius-Caesar* model via the edge-based graph under different balance parameter values. (a) The initial model. (b)-(f) Deformation results with the global balance parameters valued 0, 0.2, 0.5, 0.8, 1.0 respectively.



(a)



(b) (c) (d)

Figure 5.9: Material-constrained deformation result of the *plane* model. (a) The initial model and its specified materials. (b)-(c) Deformation results of uniformly distributed materials with global balance parameter valued 0 and 1 respectively. (d) Deformation result with the specified non-uniform materials.

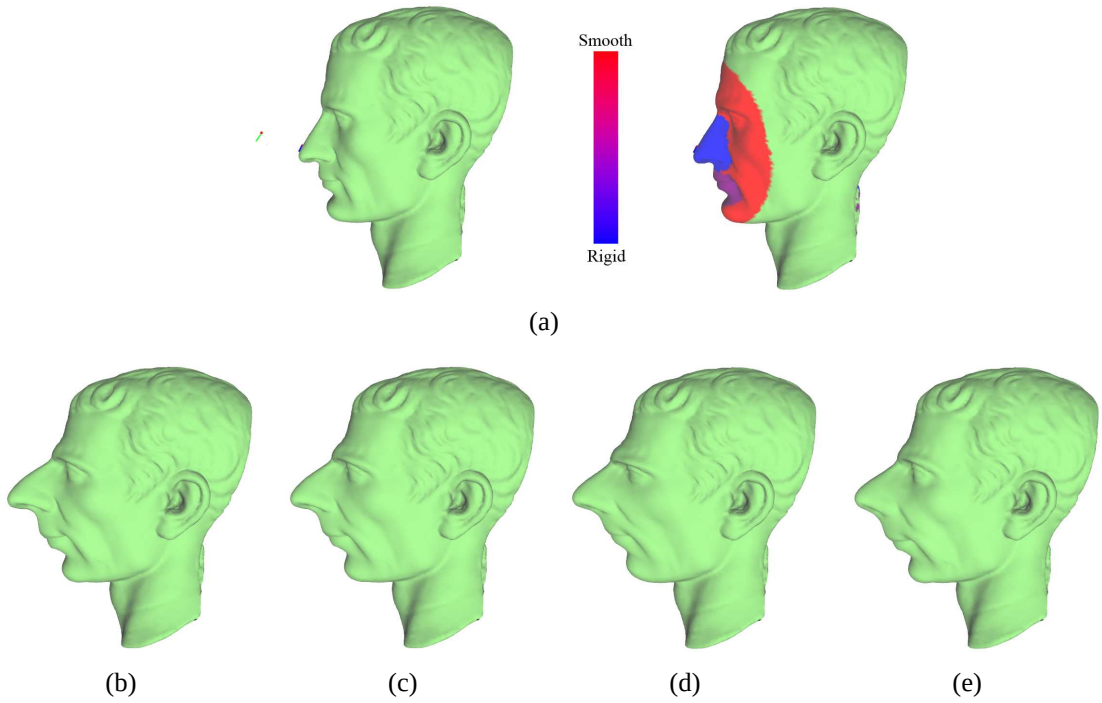


Figure 5.10: Material-constrained deformation of the *Julius-Caesar* model. (a) The initial model and its specified materials. (b)-(d) Deformation results of uniformly distributed materials with global balance parameter valued 0, 0.5 and 1 respectively. (e) Deformation result with the specified non-uniform materials.

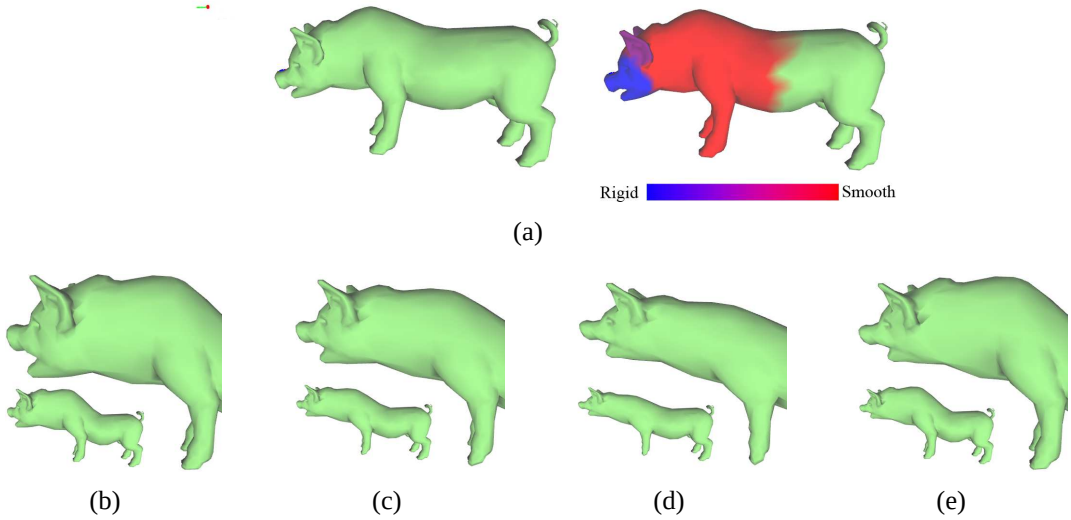


Figure 5.11: Material-constrained deformation of the *pig* model. (a) The initial model and its specified materials. (b)-(d) Deformation results of uniformly distributed materials with global balance parameter valued 0, 0.5 and 1 respectively. (e) Deformation result with the specified non-uniform materials.

5.5 Results and discussions

We tested our deformation algorithm in the hardware and software environment introduced in Section 3.4. We used the LAPACK library [11] to solve the linear equations. For the operations of the mesh deformation, we adopted a user interface presented in Chapter 3, which allows the user to sketch strokes on the mesh surface and on a reference plane as the handle curve and its deformed shape (indicated by the blue and green curves in all the examples) respectively. The ROI is also specified through sketching. The red dots in the examples indicate the static vertices. The performance of our algorithm on various examples is shown in Table 5.1.

Table 5.1: Performance of our deformation algorithm.

Model	♯ Involved vertices	♯ Involved nodes in edge-based graph	Time
plane (Figure 5.9)	812	2225	162 ms
Baby (Figure 5.2)	885	2603	181 ms
Julius-Caesar (Figure 5.8)	890	2471	171 ms
pig (Figure 5.11)	902	2599	179 ms
Buffle (Figure 5.7)	2769	8026	557 ms
Julius-Caesar (Figure 5.10)	4359	12641	982 ms
Dinosaur (Figure 5.6)	9209	27407	2192 ms

Figures 5.2 and 5.6 compare the deformation results of the flexible deformation algorithm performed directly in the primal domain and via the edge-based graph. In Figure 5.2 we can see that when λ equals 0 or 1, which means a smooth or rigid deformation is requested, calculations in the primal domain produce poor results: when $\lambda = 0$, the shape of the foot is not well preserved and when $\lambda = 1$, distortions appear around the knee part. However, the results calculated via the edge-based graph are much better. Similar observations can be found in Figure 5.6, where the tail of the dinosaur is lifted and stretched to some extent.

In Figure 5.7 we stretch the leg of the model and present different results under various λ values. The result of $\lambda = 0$ is quite smooth, but the leg becomes fatter than the initial

one, which means shape preservation is not satisfactory. The result of $\lambda = 1$ preserves the shape well. However, it lacks some smoothness. The result of $\lambda = 0.5$ seems to be the most natural and realistic one.

Sometimes the most satisfactory result cannot be easily identified, such as in the example shown in Figure 5.8, in which we deform the nose of a head model. This is reasonable because besides the requirement of local feature preservation and global smoothness, the best deformation also depends on the individual user preference. From this perspective, all these results are acceptable and the best is left for the user to choose.

It is also observed that for some models, the differences between the deformation results produced through calculating directly in the primal domain and via the edge-based graph are not quite obvious when $\lambda = 0.5$. This is because the result of $\lambda = 0.5$ can be regarded as a high-level interpolation of those of $\lambda = 0$ and $\lambda = 1$. Even the results of the pure rigid and smooth deformations done in the primal domain are far from satisfactory, interpolation of them may reduce the distortion to some extent. However, the capability of producing multiple natural and realistic results under various λ values makes the edge-based method more flexible.

Figures 5.9, 5.10 and 5.11 show the results of material-constrained mesh deformation. The deformation results for the same model with certain uniform materials are also provided for comparisons. In Figure 5.9, the material at the center part of the plane is set to be the stiffest and the material in the surrounding part to be soft. As can be seen, the deformations of these two parts approximate those of the models with corresponding material properties. In Figure 5.10, we define different materials on the nose, mouth and face, in order to let the nose deform rigidly, the face change smoothly, and the mouth change in an in-between effect. The result looks as if the nose, mouth and face are transplanted from the models with corresponding uniform materials. Similar results can be observed on the head, ear and body of the pig in Figure 5.11, which is an exaggerated deformation effect often seen in cartoons.

It can be seen that using the material-constrained deformation algorithm, we can make

the deformation more flexible to produce different natural and realistic deformation results, which is usually difficult using previous methods. Comparing with the method proposed in [129], our approach does not require the user to manually specify the local transformations of the handles and avoids the translation-insensitivity problem during deformation.

It is worth mentioning that our algorithm involves a larger linear system due to the use of the edge-based graph and thus the computation takes more time. However, in our experiments we found that the algorithm only needs up to three or four iterations to reach convergence, and since only the right hand side of the linear system needs to be updated at each iteration, the computation is still faster than those nonlinear methods.

Chapter 6

Editable Surface Reconstruction from Cross Sections with Arbitrary Orientations

This chapter presents a framework for topology-editable surface reconstruction from cross sections with arbitrary orientations, with an emphasis on how the algorithm in Chapter 4 is generalized to handle cross sections with arbitrary orientations and how the sketching technique is used in a topology editing process to increase the flexibility of the reconstruction.

6.1 Introduction

In Chapter 4 we have presented an algorithm on surface reconstruction from orthogonal cross sections, which effectively supports the progressive modeling operation in our sketching interface and avoids the limitations in the method in [101]. However in some other applications such as the 3D data recovery from medical data (CT, MRI, Ultrasound, etc.), terrain modeling and so on, the input cross sections are usually lying on planes with arbitrary orientations. In that case, the method in Chapter 4 cannot be directly used to solve this problem. Though the work [101] proposed a general solution to this problem, some

challenges still exist and remain unsolved.

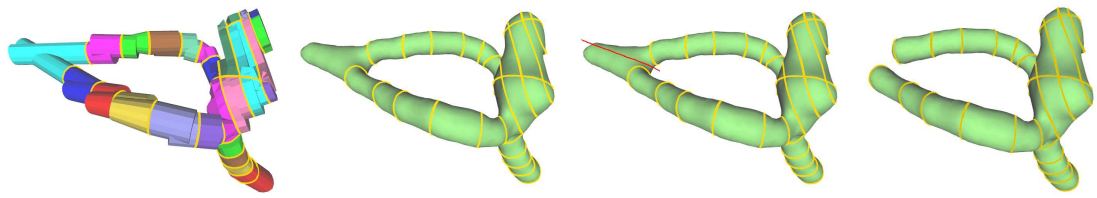
First, the reconstructed surface, which usually represents a 3D object, may contain multiple disconnected components (islands). While in most cases, an object with just one single component is often desired by the user. To avoid this problem we have proposed a strategy in Chapter 4, which is suitable for sketch-based modeling operations when the user iteratively adds new sketches close to the existing curves. However for the general-purpose surface reconstruction which reconstructs the surface from an existing set of cross sections, this problem still cannot be easily solved by using this strategy. What is worse, it may become more obvious for a large number of cross sections with arbitrary orientations and complex mutual relationship which may not occur in sketch-based modeling operations. In that case, it has been difficult to maintain both the interpolation on the input curves and regularity of the local shapes, letting alone the requirement of the single-component property.

Second, without the prior information on the topology of the target shape, the initial reconstruction result cannot be guaranteed to completely meet the user's expectation. Once it is not satisfactory, the user has to employ other editing tools to further modify it to a desired shape, which is a totally independent process of the reconstruction and may require laborious work and complicated computation.

(a) (b)

Figure 6.1: An example of surface reconstruction using Liu et al's algorithm [101]. (a) Input cross sections. (b) The reconstructed result that has several disconnected components.

To solve these problems, we propose a new surface reconstruction framework. First of all, we generalize the algorithm proposed in Chapter 4 to build a manifold surface interpolating the input cross section curves with arbitrary orientations. Generally, our method follows the same workflow to partition the whole 3D space into zones and get the final



(a) (b) (c) (d)

Figure 6.2: The proposed surface reconstruction process for the input cross sections given in Figure 6.1(a). (a) Reconstructed sub-surfaces in zones. (b) The reconstructed surface that has only one connected component. (c) Sketching a stroke (red) to indicate the local region to be split if desired. (d) The reconstructed result after splitting.

surface by stitching the sub-surfaces built inside each zone. However, we use more delicate methods for computing the sub-surface within each zone whose shape will be more complex and extending the surface components generated from some cross sections when necessary, such that regular local shapes can be produced and isolated surface parts can be effectively connected.

Moreover, we provide simple sketching tools to allow the further modification of the topology of the reconstruction result, if the user is not satisfactory with the initial shape. Different from existing topology-editing approaches, our tools make use of the intermediate calculation results obtained during the surface reconstruction process, and avoid the global analysis and processing of the 3D model from the beginning. Thus, the reconstruction and modification of a desired surface will be carried out in a consistent manner in our framework.

The most important feature of our framework is that the output surface is guaranteed to be a manifold surface having only one connected component, which is usually consistent with the user's expectation when reconstructing a 3D model from given cross sections. Since our method is a local scheme, we are able to implement the calculations of the sub-surfaces in all zones in parallel to speed up the whole surface reconstruction process. Figure 6.2 shows the reconstruction and editing of an anatomy model in our framework, with comparison to the result of a previous method [101].

The rest of this chapter is organized as follows: Section 6.2 gives an overview of our sur-

face reconstruction framework. Section 6.3 describes the details of the generalized surface reconstruction algorithm. The sketch-based topology editing of the reconstructed model, including the user interface and algorithm, is provided in Section 6.4. The experimental results and analysis are provided in Section 6.5.

6.2 Overview of the algorithm

The input in our algorithm is a set of cross sections $CS = \{cs_i \mid i = 1, ..., m\}$ lying on *arbitrary* planes $P = \{p_i \mid i = 1, ..., n\}$ in 3D space $\mathbb{R}^3$. The cross sections are closed curves represented as polylines. Similar to that in Chapter 4, the output of this algorithm is a closed 2-manifold triangular mesh M interpolating the input cross sections CS and having only one connected component.

Our method adopts the same workflow in Chapter 4 to reconstruct surfaces from cross sections. That is, we first place a large virtual bounding box outside all the cross sections. Here the size of the box is set to 2 times that of the bounding box of all points on the cross sections, and its shape may not be a cube as in Chapter 4. Then we partition it into zones $ZN = \{zn_i \mid i = 1, ..., m\}$ by using the planes that the cross sections lie on. The curves are then put onto the faces $F = \{f_{ij} \mid f_{ij} \in zn_i, j = 1, ..., n\}$ of each zone zn_i. Next we build a sub-surface which interpolates the curves on the faces of each zone and stitch all the pieces of sub-surfaces together to form a complete surface. Finally we improve the quality of the mesh surface through iterative refinement and smoothing.

Similar to the algorithm in Chapter 4, we use the CSG-based method to generate the sub-surface which interpolates curves on the zone faces, such that a manifold surface with regular local shapes can be reconstructed. We also extend the potentially isolated surface components by adding some virtual cross sections, to reduce the possibility of the existence of isolated surface components. Comparing to the previous method [101], our approach does not rely on any geometric agency and can join adjacent surface parts flexibly. What is more, the information obtained during this process can be retained and further used for the

computation of topology editing of the reconstructed model, such that complicated analysis and computation of the global surface in topology editing operations can be avoided.

In next two sections, we describe the surface reconstruction algorithm and the sketch-based topology editing approach in detail.

6.3 Surface reconstruction

A basic goal for sub-surface reconstruction is to make the result surface totally lie within the zone and interpolate the input curves on its faces. We do not use any geometric agency (for example, the medial axis plane in [101]) whose computation is sensitive to the shape of the zone to avoid creating artifacts. Meanwhile, it would be better if the scheme is re-configurable, to make the topology of the local surface more flexible to adjust when the initial result is not satisfactory.

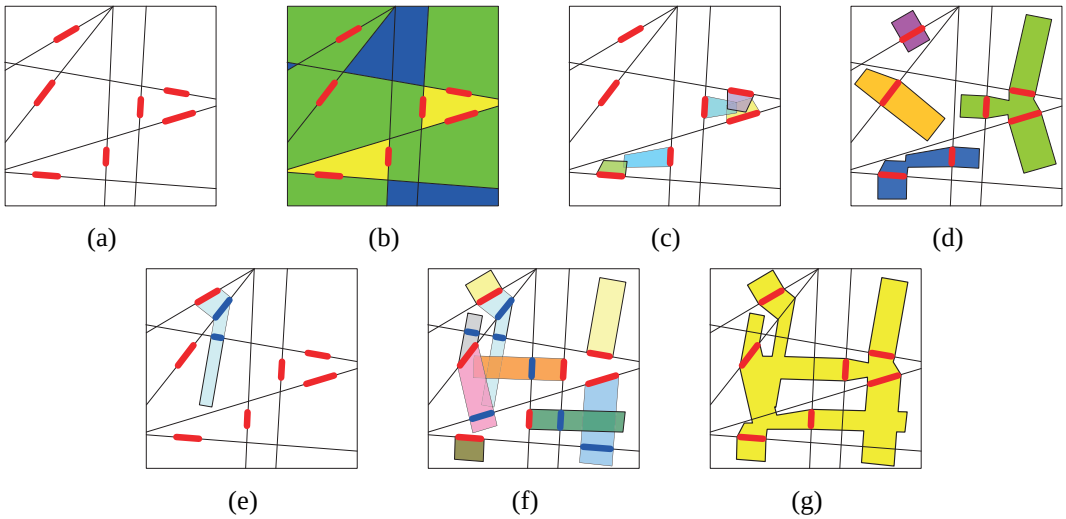


Figure 6.3: 2D illustration of our surface reconstruction algorithm. (a) Input segments (red, corresponding to the 3D cross sections) and partitioned zones. (b) The *empty* zones (blue), *end* zones (green) and *body* zones (yellow). (c) Generating polygons (corresponding to the frusta) in *body* zones. (d) Generating polygons in the *end* zones using the same way as the reconstruction in the *body* zones. The reconstruction result contains multiple components. (e) Generating extended polygons for an *end* zone. The blue segments correspond to the virtual cross sections we built. (f) Generating polygons for all *end* zones. (g) The reconstruction result.

To achieve these goals, we first use the strategy in Chapter 4 to partition zones into three

types: the *empty* zone which has no faces containing cross sections, the *end* zone with only one face containing cross sections, and the *body* zone with two or more faces containing cross sections. A 2D illustration of the partition result and zone types can be found in Figure 6.3, in which the segments on the lines correspond to the cross sections on the zone faces.

However, different from the case in Chapter 4, the shape of each zone is generally not a cuboid. As a result, the method on generating the cylinders cannot be simply used since it is difficult to ensure that the built cylinders lie within the zone. So here for each cross section cs_p in zone zn_i, we build a frustum, instead of a cylinder, as the generated sub-surface interpolating cs_p in zn_i and then compute the union of these frusta. The surface of the result solid is then taken as the sub-surface computed in zn_i, and all the pieces of sub-surfaces are stitched together to form the global surface.

6.3.1 Sub-surface reconstruction for a *body* zone

In a *body* zone, each cross section may either intersect with another cross section on a different face or not. Depending on this intersection relationship, different strategies are adopted to create the frustum from each cross section.

Case 1: For a cross section cs_p on face f_{ij} which does not intersect with other cross sections in zone zn_i (see Figure 6.4(a)), we need to build a frustum fr_p taking the 2D region enclosed by cs_p as the bottom fr_p^b. The initial top fr_p^t of fr_p is obtained by translating fr_p^b along the direction orthogonal to f_{ij} and pointing towards the inside of zn_i. To get the distance of this translation (i.e. the initial height value h_p for the frustum), we compute a ray starting from the center of fr_p^b and pointing towards the inside of zn_i with the directions orthogonal to f_{ij}. The ray will intersect with the other faces of zn_i and the nearest intersection point to the center of fr_p^b is selected. Suppose the distance between these two points is d_p, then the height will be computed as:

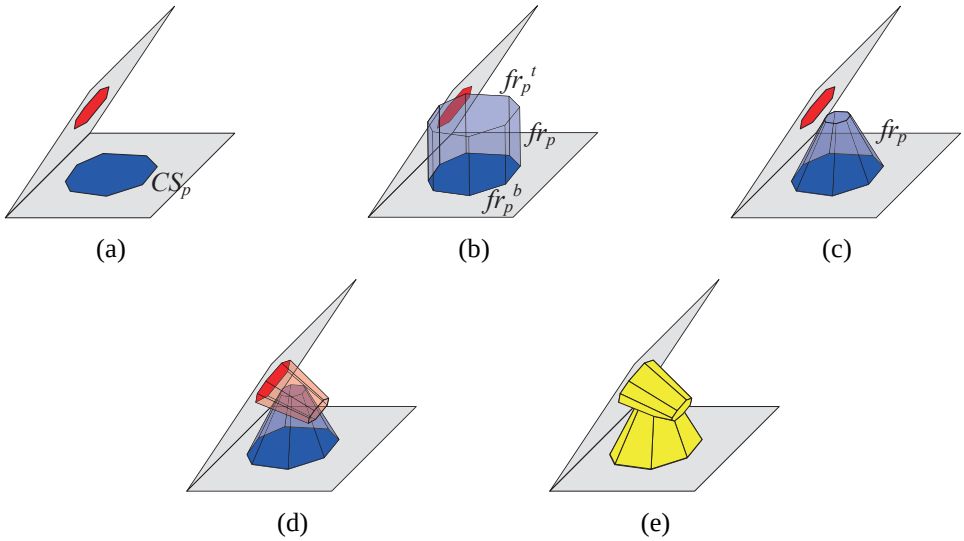


Figure 6.4: Illustration of sub-surface reconstruction for non-intersected cross sections in a *body* zone. (a) Input cross sections. (b) Generate initial frustum fr_p from one cross section. (c) Adjust the top fr_p^t of the frustum to make it lie within the zone. (d)Generate another frustum. (e) Compute the union of the two frusta.

$$h_p = d_p - \epsilon, \tag{6.1}$$

where ϵ is a small positive number to make sure that the center of fr_p^t lie within zn_i. Then if there are any points on fr_p^t lying outside zn_i (see Figure 6.4(b)), we keep shrinking fr_p^t towards its center until fr_p^t locates totally inside of zn_i or a predefined shrinkage ratio is reached (see Figure 6.4(c)). If the latter happens, we iteratively decrease the height value h_p and re-compute fr_p^t in this method until fr_p totally lies inside zn_i. We do not allow fr_p^t to shrink to a single point to avoid making fr_p a cone which may cause numerical issues in the subsequent union calculation. It can be seen that this method is simple and quick to construct a frustum which is guaranteed to lie totally within the zone and interpolate the cross section.

Case 2: When a cross section cs_p on face f_{ij} intersects with another cross section cs_q on the neighbor face f_{ik} of f_{ij} in zone zn_i (see Figure 6.5(a)), the goal of the sub-surface reconstruction is to make sure that fr_p and fr_q (frusta generated from cs_p and cs_q respectively) lie within zn_i and intersect with each other on more than the common segment

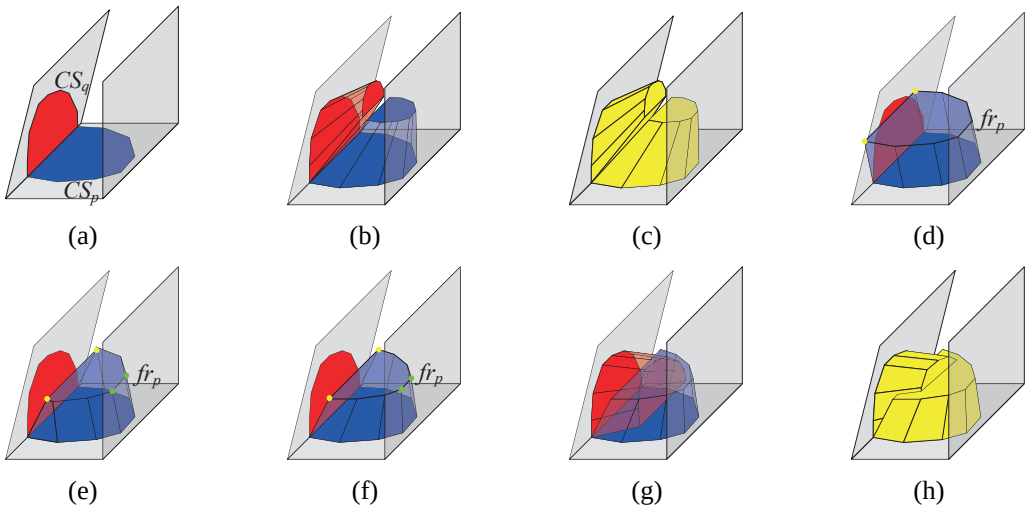


Figure 6.5: Illustration of sub-surface reconstruction for intersected cross sections in a *body* zone. (a) Input cross sections. (b) Generate the frusta by shrinking the top faces. (c) Non-manifold sub-surface caused by the over-shrinkage of the top of each frustum. (d) Translate the bottom fr_p^b to get the initial frustum fr_p. The yellow dots represent the points outside the zone (i.e. PT_f in Eq 6.2). (e) Move the outside points into the zone. The green dots are the points near the other faces of the zone (i.e. PT_o in Eq 6.2). (f) Deform the boundary of the frustum top by keeping the yellow and green points fixed, to get a valid frustum. (g) Generate another frustum. (h) Compute the union of the two frusta.

of cs_p and cs_q, such that the surface of their union is a closed manifold that interpolates both the two cross sections. In that case, the bottom fr_p^b is simply taken as the region enclosed by cs_p and the edges of f_{ij}, but the computation of the top fr_p^t will be more involved. Especially when the dihedral angle between f_{ij} and f_{ik} is equal to or less than $\pi/2$, simple translation of fr_p^b along the orthogonal direction will make cs_q covered by fr_p^t (see Figure 6.5(d)) and the union result will not interpolate cs_q, while over-shrinking fr_p^t like the way we handle Case 1 may make the union result a non-manifold surface (see Figure 6.5(b) and 6.5(c)).

Here we first adopt the same method in handling Case 1 to get the initial height value h_p and the top fr_p^t of fr_p. Especially, we select the points on fr_p^b whose distances to f_{ik} are smaller than a given threshold and mark their corresponding points PT_f on fr_p^t (the yellow points in Figure 6.5(d)). These points are supposed to stay near f_{ik} inside zn_i to make the union of fr_p and fr_p be a valid solid whose surface is a manifold. Then we check if there

exist any points on fr_p^t lying outside zn_i. If yes, we translate them towards the center of fr_p^t until they just get inside of zn_i (see Figure 6.5(e)). Next, to preserve the original shape of fr_p^t and avoid undesired artifacts caused by the translation of these points, we launch a deformation process to deform the boundary curve PT_b of fr_p^t. Specifically, we try to fix the positions of PT_f and preserve the Laplacian vectors which are used to represent the local shape of the original curve. The Laplacian vector δ_i at each boundary points pt_i of the initially obtained fr_p^t can be calculated by using Eq. 3.2.

We also fix the positions of points PT_o which stay near the other faces of zn_i (the green points in Figure 6.5(e)), to prevent them from getting out of zn_i after the deformation. So the total energy to minimize for the deformation is:

$$\begin{aligned} E(PT_b') &= \sum_{pt_i \in PT_b} ||L(pt_i') - T_i\delta_i||^2 + \\ &\quad \sum_{pt_i \in PT_f} ||pt_i' - pt_i||^2 + \\ &\quad \sum_{pt_i \in PT_o} ||pt_i' - pt_i||^2, \end{aligned} \tag{6.2}$$

where pt_i and pt_i' represent the original and deformed point positions. T_i denotes the transformation matrix of the Laplacian vector. This curve deformation problem can then be solved by using the same method we have introduced in Section 3.3.2.

Although we tried to anchor the points near the boundary faces of zn_i, there may still exist some other points staying outside zn_i after the deformation. In that case, these points are then translated towards the center of the deformed curve PT_b until they come into the inside of zn_i and the set PT_o will be updated accordingly. After a few iterations of using Eq. 6.2, all points on PT_b will be inside zn_i and valid fr_p can then be obtained (see Figure 6.5(f)). This method also works when cs_p intersects with multiple cross sections.

6.3.2 Sub-surface reconstruction for an *end* zone

It has been introduced in Chapter 4 that without extension, the sub-surfaces generated in the *end* zones may probably become isolated and the whole surface will contain multiple components. This may also happen for cross sections with arbitrary orientations (see Figure 6.3(d)). However, since the shape of each zone may not be a cuboid, the neighbor zone to extend the frustum to should be carefully selected. The simple method of extending in the orthogonal direction in Chapter 4 needs to be improved.

Here we adopt a new strategy to extend the surface parts generated from cross sections in the *end* zones to non-end zones by adding some virtual cross sections such that their possibility of becoming disconnected surface parts will be minimized.

(a) (b)

Figure 6.6: Illustration of sub-surface reconstruction in an *end* zone. (a) The input cross section cs_p on face f_{ij} is projected onto the other faces in the orthogonal direction of f_{ij} and the face f_{ik} with the largest projection area is selected. (b) A virtual cross section cs_v is obtained by projecting cs_p onto f_{ik} and the frustum fr_p is built by taking cs_p and cs_v as the bottom and top.

Specifically, for each cross sections cs_p on face f_{ij} in an *end* zone zn_i, we build a frustum fr_p taking the 2D region enclosed by cs_p as the bottom fr_p^b. Different from the methods in Section 6.3.1, we check if the top fr_p^t of fr_p can be built on another face f_{ik} of the zone, such that the surface components generated from cs_p can be easily extended to other zones. To do that, we project fr_p^b onto other faces of the zone in the direction orthogonal to f_{ij}, and the face with the largest projection area will be selected as f_{ik} (see Figure 6.6(a)). If f_{ik} belongs to the bounding box, then the extension will be considered as impossible and the frustum will be built just in the method we handle Case 1 in a *body* zone, and this frustum will be treated as the surface component generated from cs_p.

If f_{ik} does not belong to the bounding box, the extension is regarded as possible. In that case, to get the initial top fr_p^t, fr_p^b will be projected onto the plane that f_{ik} lies on along the direction orthogonal to the plane. If there are any points of fr_p^t lying outside f_{ik}, we just shrink fr_p^t towards its center until fr_p^t is completely bounded by f_{ik}. By connecting the corresponding points on fr_p^b and fr_p^t, the frustum in zn_i can be built (see Figure 6.6(b)). Next, we take the boundary curve of fr_p^t as a virtual cross section cs_v and continue building a new frustum from cs_v in the other zone zn_k incident to face f_{ik}, by using the same method as above. This process continues until the new face for projecting the virtual cross section belongs to the bounding box, or the next zone is a *body* zone, and the method of handling Case 1 in a *body* zone will be used to generate the last frustum. Finally, all the connected frusta which are created during this process will be taken as the surface components generated from the cross section cs_p.

By adding the virtual cross sections and extending the frusta, the possibly isolated surface components in the *end* zones can be effectively connected to other components in most cases, since there is a very low possibility that all their extensions end up at the boundary faces. Thus, the possibility of existence of multiple components in a reconstructed model will be minimized. Since this process starts from each *end* zone simultaneously and independently, the reconstruction result will be unique.

It has been mentioned in Chapter 4 that another reason that causes the multi-component problem is that the sub-surfaces generated from different cross sections in a *body* zone may not intersect with each other. In this reconstruction scheme, it can be solved by changing the shape and orientation of the frusta to force them intersect, which is easy to do in a convex zone. However, we do not make that compulsory since it may cause irregular local shapes and undesired connections. As we will show in the next section, we give this freedom to the users to allow them join these components by sketching a simple curve if needed.

6.3.3 Global surface reconstruction

Similar to Chapter 4, we use the algorithm proposed in [91] to compute the union of the generated frusta. Since the frusta are simple shapes, this method is fast and robust enough for our application. The calculation of the global surface M_{init} can be expressed as:

$$M_{init} = \sum_{i=1}^{m} (\bigcup_{j=1}^{n} fr_{ij}) \tag{6.3}$$

in which fr_{ij} refers to the jth frustum built in zone zn_i, assuming there are totally m zones and n frusta in each zone. After the computation, all fr_{ij} and their unions will be preserved for the further edit operation if necessary, which will be introduced in detail in Section 6.4. Finally, we iteratively refine and smooth the mesh surface using the same algorithms adopted in Chapter 4.

It can be seen that in our approach, the reconstruction result directly depends on the primitives generated in each zone. This makes our method free of computing any geometric agency and insensitive to the shape of the zone. What's more, the shape of the primitive is adjustable and different shapes may lead to different reconstruction results, which provides the flexibility for the further editing of the initial shape.

Since our algorithm is a local scheme, the computation of the sub-surface in one zone is independent of that in others. So the reconstruction in each zone can be done in parallel and the calculation can be accelerated.

6.4 Topology editing

After the reconstruction process, a generally satisfactory result can be obtained. However, since no prior information on the topology of the target model is provided before the reconstruction, the shape of some local surface, especially the connectivity may not fully meet the user's expectation though the global shape is guaranteed to be reasonable. In that case, further editing of the model is often needed.

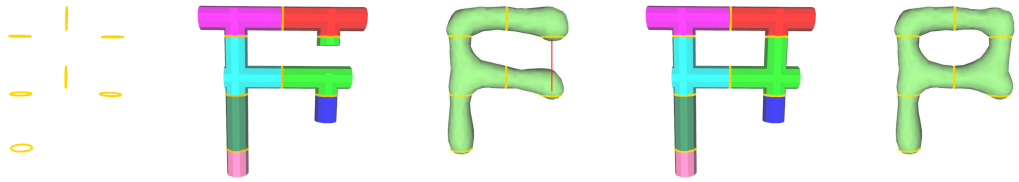
To avoid tedious traditional operations on mesh vertices or faces, there have been some sketch-based topology editing methods proposed [64, 74, 77], which modify a model by requiring the user to provide hints of the desired topology through sketches and analyzing the global shape of the model. Although these global methods can be used to edit the reconstructed shape, they seem complicated for small local modifications, on both the user interface and algorithm. The main reason behind this inconvenience is that existing methods consider the surface reconstruction and editing as two independent operations, which means no extra information beyond the model shape obtained in the former process is utilized in the latter one.

To provide the user with a light-weight tool for quick modification of the topology of the local surface, we propose a new approach which makes use of not only the user sketches and the model but also the intermediate information obtained during reconstruction to assist the computation of the further topology editing. More precisely, the intermediate information here refers to the frusta generated from the cross sections in each zone. Since the final global surface is formed from these basic primitives, local surface modification can be simply done by changing the shape and connectivity of the frusta at a specific region.

There are two operations provided in the editing tool: join and split. The former is used to join the disconnected surface parts and the latter is for separating a single component into multiple parts. Given the reconstructed model, the user can first adjust the viewpoint and then sketch a simple 2D curve to implement the two editing operations.

6.4.1 The join operation

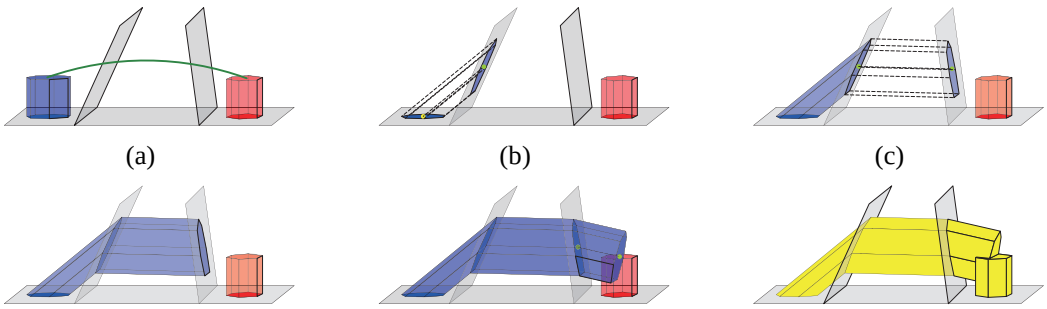
In the join operation, the user is allowed to sketch a curve to select the two surface parts to be connected (see Figure 6.7(c)). In the background, the two frusta that are nearest to the camera among all those selected by the curve will be selected (the frusta with light green color in Figure 6.7(b)). If they are in the same zone, then we compute the centers of the tops of the two selected frusta, and deform the first selected frustum by moving its top along the



(a) (b) (c) (d) (e)

Figure 6.7: An example of the join operation in sketch-based topology editing. (a) Input cross sections. (b) Computed frusta in zones. (c) Sketch a curve (red) on the reconstruction result to join surface parts. (d) The corresponding frustum (with light green color) is deformed. (e) Editing result.

vector pointing from its center to that of the second selected one until they get intersected (see Figure 6.7(d)). Finally the union of the frusta within this zone is recomputed and the global surface is updated (see Figure 6.7(e)). When multiple frusta are selected in a zone, the last selected one will keep fixed and the other ones will be deformed in the way to intersect with it.



(a) (b) (c)

(d) (e) (f)

Figure 6.8: Illustration of the join of surface parts in different zones. (a) Sketch a curve (green) to select the two frusta. (b) Re-calculate the top of the first selected frustum by projecting its bottom onto the common face with the neighbor zone. (c)-(d) Extend the frustum. (e) Build the extended frustum in the zone containing the second selected frustum. (f) Editing result.

We also allow the join of surface parts in different zones. In that case, the two frusta and the zones they belong to will be firstly selected (see Figure 6.8(a)). Then we need to find a series of adjacent zones connecting these two zones. To do that, we start this searching process from the first selected zone, to find its neighboring zones whose 2D projections on the screen contain the points of the sketched 2D curve. This searching process continues

from these candidate zones until the last selected zone is reached. A set of zones connecting the first and last selected zones, as well as the common faces incident to each pair of zones can thus be obtained. Next, we re-calculate the first selected frustum by projecting its bottom onto the first calculated common face to get the new top (see Figure 6.8(b)) and continue extending the frustum across the selected zones in this way until the last zone is reached (see Figure 6.8(c) and 6.8(d)). Note that this method is quite similar to that we used in Section 6.3.2. However, since the dihedral angle between the plane of the face f_t to project and that of the face f_b containing the bottom of the frustum may not be acute, projecting in the orthogonal direction of f_t will not work in that case. So here we project the bottom of each frustum along the direction pointing from its center to that of f_t, to get the frustum top. The top of the extended frustum in the last zone will be calculated by translating the bottom along the vector pointing from its center to that of the top of the second selected frustum (see Figure 6.8(e)) such that the two frustum can get intersected (see Figure 6.8(f)).

6.4.2 The split operation

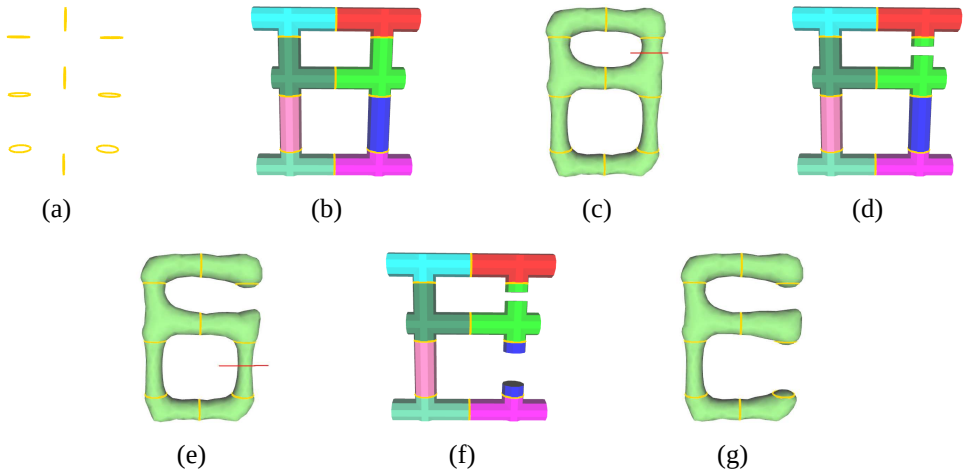
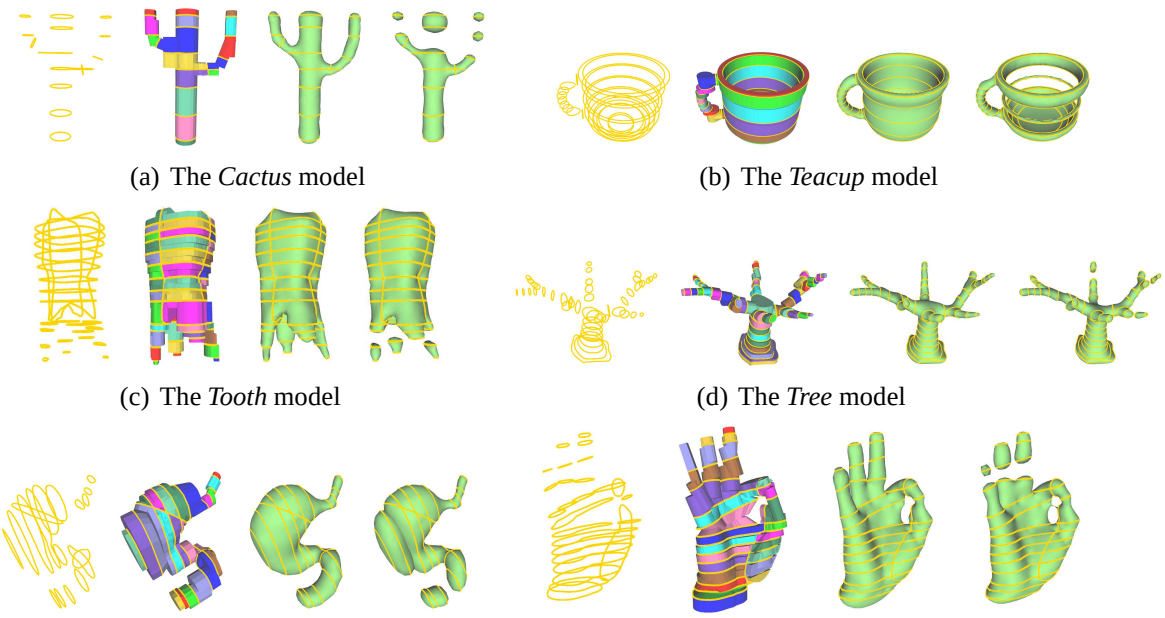


Figure 6.9: An example of the split operation in sketch-based topology editing. (a) Input cross sections. (b) Computed frusta in zones. (c) Sketch a curve (red) on the reconstruction result to split local surface. (d) The corresponding frusta (with light green color) are shrunk to be separated. (e) Sketch another split curve on the updated model. (f) The corresponding blue frusta are shrunk. (g) Editing result.

The split operation applies to the local components that are formed from connected surface parts generated from two or more cross sections within the same zone. If the component to split is generated from one single cross section in a zone, splitting it will produce an extra isolated surface component which does not interpolate any cross section curves in this zone. To split a local surface component, the user needs to first adjust the viewpoint to make it foremost and then sketch a cutting curve passing through it(see Figure 6.9(c) and 6.9(e)). Next all the connected frusta that contribute to the selected surface part are selected and shrunk towards their bottoms until they do not intersect (The light green frusta in Figure 6.9(d) and the blue frusta in Figure 6.9(f)). This guarantees that the surface parts obtained after the union operation will be separated. The global surface can then be updated.

It can be seen that by making use of the generated frusta obtained in the reconstruction process in the local topology editing operations, our approach builds a bridge that efficiently connects the two originally independent operations, such that a desired shape can be produced for the user through simple sketches and local computation when the initial reconstruction result is not satisfactory. The final surface produced by our surface reconstruction framework will have regular local shapes and be composed of only one component.

It should be mentioned that the two editing operations can also be incorporated into the surface reconstruction step, by adopting strategies to set the shape of the generated frusta. For example, we can force each frustum to be connected with others, such that the output surface after the first reconstruction step will be guaranteed to have a single component. However, this compulsive integration may produce irregular local shapes and unexpected connectivity of different surface parts. Thus we just try to produce a valid model with regular shapes in the first step and leave the adjustment of the local topology to the user.



(a) The *Cactus* model (b) The *Teacup* model

(c) The *Tooth* model (d) The *Tree* model

(e) The *Stomach* model (f) The *Hand* model

Figure 6.10: Results of our surface reconstruction algorithm. In each example, the input cross sections, the generated sub-surfaces in zones, the reconstruction result of our algorithm, and the result of the algorithm of [101] are shown in order.

6.5 Results and discussions

We have demonstrated our surface reconstruction algorithm on both the bio-medical and synthetic models in Figure 6.10. Cross sections on arbitrary planes with complex relationship were taken as the input to test the universality and robustness of the algorithm. These cross sections are computed from the models provided in the INRIA Gamma team research database [7]. For comparison, we also present the results generated using the algorithm of [101], which represents the state-of-the-art and is closely related to our work.

By extending the surface components generated in the *end* zones and adjusting the shape of the generated frusta to connect isolated surface parts, our algorithm can produce a manifold surface with only one connected component. All the examples in Figure 6.10 have demonstrated this advantage of our scheme over the existing methods which use the traditional partition approach. This is more obvious on the shapes with multiple branches than those with topology equivalent to a simple sphere. This is because in the former case, the distribution and relationship of cross sections are more complicated and faces which may

separate the connections of different surface components are more likely to exist.

Since our sub-surface reconstruction is independent of any geometric agency (eg, the medial axis plane in [101]), topological noise caused by the irregularity of the agency in previous methods can be eliminated. Such an example can be seen in Figure 6.10(f), in which undesired artifacts are created in the local surface between the thumb and palm in the result of [101] but are avoided in our result.

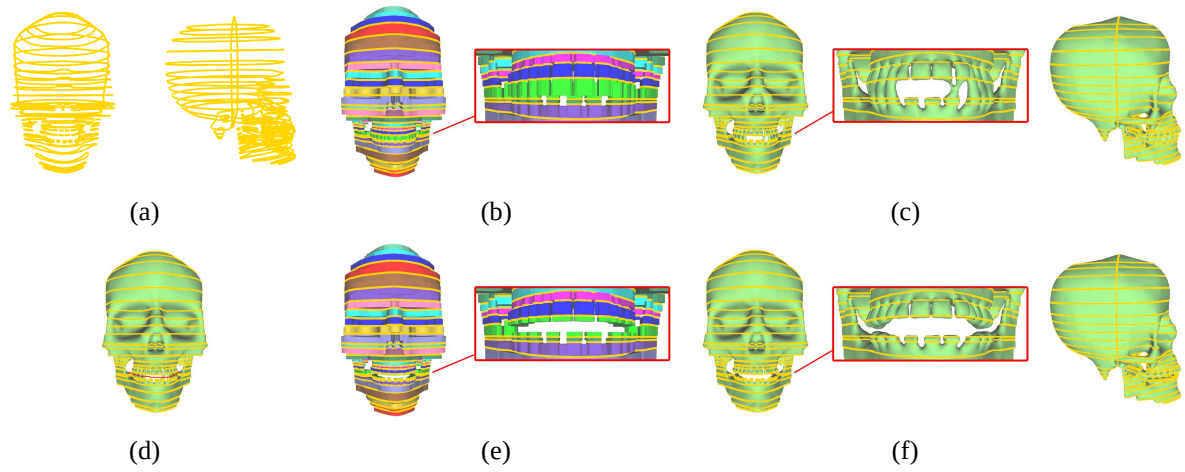


Figure 6.11: An example of the reconstruction and editing of the *skull* model. (a) Input cross sections. (b) Generated sub-surfaces in zones. (c) Reconstruction result. (d) Sketch a curve (red) to split the local surface. (e) Updated sub-surfaces in zones. (f) Editing result.

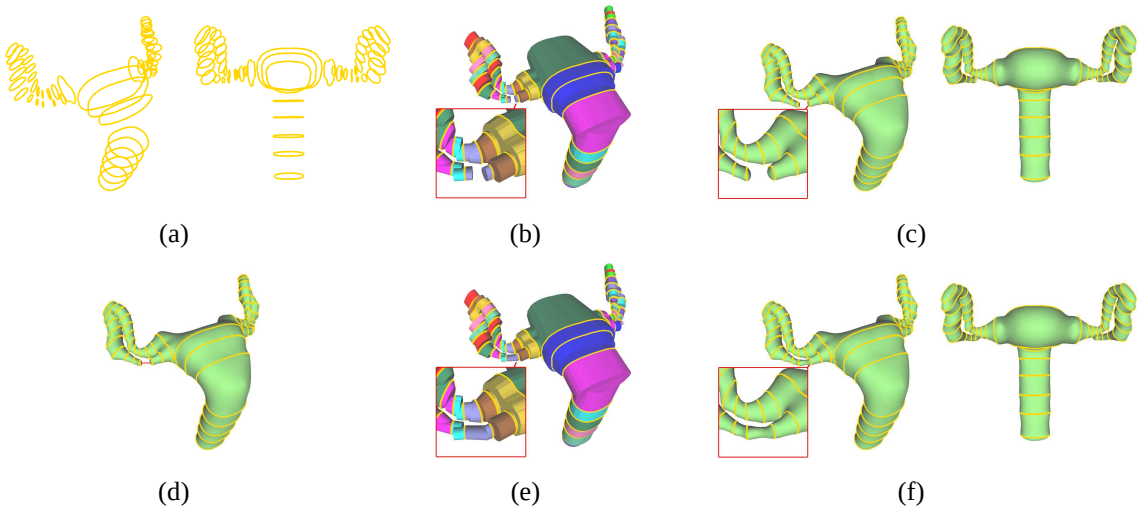


Figure 6.12: An example of the reconstruction and editing of the *ovaries* model. (a) Input cross sections. (b) Generated sub-surfaces in zones. (c) Reconstruction result. (d) Sketch a curve (red) to join the local surface. (e) Updated sub-surfaces in zones. (f) Editing result.

Figure 6.11 shows an example on the reconstruction and editing of a more complicated model. Note that although the global shape of the skull has been satisfactory after the reconstruction, the upper and lower teeth are stitched, which is not expected. By sketching a simple stroke to select the surface components to split and changing the shapes of the generated frusta during the reconstruction process, the local surface can be easily updated as desired, while for other sketch-based topology editing tools more complicated user input and global computation are inevitable. Other editing examples can be found in Figures 6.2 and 6.12.

Now we briefly analyze the time complexity for our reconstruction algorithm. Getting the initial partition result for cross sections on n_p planes will take $O(n_p^3)$ time as in [24] and [101]. Supposing there are totally m points on the cross sections in a zone, there will be $2m$ vertices and $2 + m$ faces on all the generated frusta in this zone according to our method of constructing each frustum. In that case, computing the union of the frusta in each zone will take at most $O(m^2)$ time, according to the demonstration in [91]. This step is more time-consuming than the projection-based methods. However, since not all the frusta are intersected, and the computation within each zone can be implemented in parallel, this time can be reduced a lot.

Our current implementation of the surface reconstruction and editing algorithm was written in C++ on the Windows platform. To accelerate the sub-surface reconstruction in each zone, we used the OpenMP library [124] to do the calculation in parallel. We tested our algorithm on an machine with Dual Core Intel Xeon 2.27GHz CPU and 2G RAM. The performance of our approach on the examples is shown in Table 6.1.

Table 6.1: Performance of our surface reconstruction algorithm.

Model	♯ Input cross sections	♯ Input points	♯ Zones after partition	♯ Output triangles	Initial surface reconstruction	Mesh improvement
cactus (Figure 6.10(a))	14	251	37	3950	1223 ms	1719 ms
stomach (Figure 6.10(e))	18	413	73	4710	1929 ms	2141 ms
anatomy (Figure 6.2)	23	524	856	7548	3484 ms	3250 ms
tooth (Figure 6.10(c))	24	760	56	9054	4773 ms	9359 ms
tree (Figure 6.10(d))	45	829	3202	10284	5136 ms	4672 ms
hand (Figure 6.10(f))	25	1094	67	14834	6711 ms	17406 ms
teacup (Figure 6.10(b))	24	1486	28	23862	8383 ms	15796 ms
ovaries (Figure 6.12)	50	2638	1941	43944	25582 ms	44859 ms
skull (Figure 6.11)	103	5915	50	72432	45046 ms	39343 ms

Chapter 7

Conclusions

7.1 Summary

This thesis has described our research of developing intuitive interfaces and effective algorithms for sketch-based 3D modeling and reconstruction. Our aim is to develop modeling techniques and tools allowing the user to effectively create, edit, and manipulate 3D models using sketch-based interface. To this end, various techniques for surface reconstruction and editing have been investigated. The analysis on user behavior, psychology and intrinsic properties of 3D shapes have been employed to guide the development of techniques for better and high-level manipulation. We have established a framework for sketch-based modeling and developed supporting algorithms for creation and editing operations in the sketching interface. We also present a solution to the problem of surface reconstruction from cross sections. The goal of our research has been fulfilled. In particular, our work of the research includes:

We present a reference plane assisted sketching interface for freeform shape design. We propose some rules for regularizing and interpreting the user input when sketching the shape of the 3D model to be created. These rules consider both the semantic meaning of the sketched strokes and human psychology and partially eliminate the ambiguity of understanding the user input sketches. We propose to use reference planes to assist the

creation and editing functions, such that the location and orientation of strokes in 3D space become meaningful and intuitive. These reference planes are created automatically during the process of sketching and help make the sketch-based modeling process intuitive and easy to use.

We introduce a progressive approach for modeling, which allows the user to create a 3D model by iteratively sketching cross section curves, taking both the up-to-date reconstructed model and the existing curves as references. In this way, perception ambiguities on the sketched 3D shape can be eliminated to a large extent and the sketching function in sketch-based modeling systems is enhanced to allow the creation of complex 3D models with arbitrary topologies. To effectively support this operation, we develop a new surface reconstruction algorithm which enables gradual shape updating and produces models with a singly connected component when the user iteratively adds new sketches.

To better support the sculpting function in our sketching interface, we propose an edge-based flexible mesh deformation algorithm. An edge-based graph for a triangular mesh is proposed to evaluate more points on the mesh surface without explicit re-meshing or re-sampling and deliver more accurate computation result. Based on the edge-based graph, a mixed deformation model, which considers both the changes of the first order and the second order differential quantities, has been proposed. Various deformation effects between local shape preservation and global smoothness can thus be produced by tuning a simple scalar parameter. The scalar parameter can also be applied to local regions of the mesh to simulate the deformation behavior of real-world objects with non-uniform materials. Experimental results have demonstrated the effectiveness and flexibility of our approach.

We also extend our techniques to provide a solution to the challenging problem of surface reconstruction from cross sections with arbitrary orientations. The solution consists of two modules. The first one is generalizing the method in our progressive modeling to generate sub-surfaces for irregular zones and to extend some sub-surfaces to connect possibly isolated surface components. The second one is further editing of local topology of the initially generated model. By utilizing the intermediate results obtained during the sur-

face reconstruction process instead of from scratch, we provide the user a simple sketching tool to quickly edit the local topology of the generated model if the initial shape is not satisfactory. As a result, the final output model of our surface reconstruction is guaranteed to be a manifold surface having only one connected component, which is usually difficult to achieve in previous methods. Various examples have demonstrated the usefulness and efficiency of our approach.

7.2 Future directions

In our sketching interface, we have provided users planes as references for mapping the 2D sketches into 3D, such that the user input will become intuitive and meaningful. There exist other types of geometric objects such as sphere and cylinder, which can be taken as references, to make the user interface intuitive and flexible. Nevertheless, with the introduction of more reference types, the intelligent selection of references may become a challenge. This thus requires careful evaluation on the trade-off since each type of references has its own pros and cons. Moreover, the combination of different types of references may be considered.

Currently the input in the sketching function of our system is allowed to be the cross section curve, which can be quickly sketched and provides an intuitive and accurate description of a 3D shape. However when an artist/designer sketches a 3D model, other strokes which depict the features, such as the hatching, shading, and suggestive contours [45] are also used. These shape descriptors may also be incorporated into our sketching function. This involves not only how to interpret the user sketches, but also how the progressive surface reconstruction algorithm can be improved to have more types of input curves besides the cross sections.

For the mesh deformation algorithm, most existing works tried to maintain various local and low-level geometric properties of the mesh, while some recent works [57, 170] have shifted the focus to high-level shape editing which preserves the structural and se-

mantic shape features of man-made objects. However for free-form objects, this remains a big challenge. There have been various feature descriptions proposed, such as ridges and valleys [119], and suggestive contours [45], though they usually do not have the regular relationships as the features in the man-made objects. After these features are extracted and analyzed, the corresponding deformation algorithms on preserving these features should be explored.

For the problem of surface reconstruction from cross section curves, currently the topology of the output model is initially decided by the relative positions of the cross sections. Though we provide a sketching tool for the further modification of the topology, it still seems troublesome when the target one is far more different from the initial shape. In that case, some other descriptors which suggest the target topology can also be included as the input besides the cross sections. For example, it will be interesting to allow the user to sketch the skeleton of the target shape which could suggest the desired topology. Meanwhile, template models which have similar topologies can also be utilized to guide the reconstruction. Correspondingly, our surface reconstruction algorithms should be modified to incorporate more input information.

Moreover, the surface reconstruction algorithm can also be extended to produce subdivision surfaces which have higher quality and scalability than mesh surfaces. The output in that case will be a coarse control mesh, whose limit surface under subdivision rules interpolates or approximates the input cross sections. The challenges will include how to build the initial control mesh and how to reduce the number of extraordinary points which are not preferred in subdivision surfaces. This technique may also be used to help create high quality 3D models, which is a challenging problem in sketch-based modeling.

References

[1] Archipelis designer. Available from: <http://www.archipelis.com/>.

[2] Autodesk 3ds max. Available from: <http://usa.autodesk.com/3ds-max/>.

[3] Autodesk maya. Available from: <http://usa.autodesk.com/maya/>.

[4] Carvecsg. Available from: <http://carve-csg.com/>.

[5] E-frontier sunny3d. Available from: <http://graphic.e-frontier.co.jp/sunny3d>.

[6] Google sketchup. Available from: <http://sketchup.google.com>.

[7] Inria gamma team research database. Available from: <http://www-roc.inria.fr/gamma/gamma/disclaimer.php>.

[8] F. Abbasinejad, P. Joshi, and N. Amenta. Surface patches from unorganized space curves. In *SoCG '12: Proceedings of the 2012 Symposuim on Computational Geometry*, pages 417–418, 2012.

[9] M. Alexa. Local control for mesh morphing. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications 2001*, pages 2–9, 2001.

[10] A. Alexe, V. Gaildrat, and L. Barthe. Interactive modelling from sketches using spherical implicit functions. In *AFRIGRAPH '04: Proceedings of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 25–34, 2004.

[11] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK's user's guide*. Society for Industrial and Applied Mathematics, 1992.

[12] A. Andre, S. Saito, and M. Nakajima. Crosssketch: freeform surface modeling with details. In *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 45–52, 2007.

[13] O. Au, H. Fu, C. Tai, and D. Cohen-Or. Handle-aware isolines for scalable shape editing. *ACM Transactions on Graphics(Proc. SIGGRAPH 2007)*, 26(3), 2007.

[14] O. Au, C. Tai, L. Liu, and H. Fu. Dual Laplacian editing for meshes. *IEEE Transaction on Visualization and Computer Graphics*, 12(3):386–395, 2006.

[15] S. Bae, R. Balakrishnan, and K. Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *UIST '08: Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, pages 151–160, 2008.

[16] S. Bae, R. Balakrishnan, and K. Singh. Everybodylovessketch: 3d sketching for a broader audience. In *UIST '09: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, pages 59–68, 2009.

[17] C. Bajaj, E. Coyle, and K.-M. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Models and Image Processing*, 58(6):524–543, 1996.

[18] G. Barequet, M. Goodrich, A. Levi-Steiner, and D. Steiner. Contour interpolation by straight skeletons. *Graphical Models*, 66(4):245 – 260, 2004.

[19] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding*, 63(2):251 – 272, 1996.

[20] G. Barequet and A. Vaxman. Nonlinear interpolation between slices. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 97–107, 2007.

[21] M. Bein, S. Havemann, A. Stork, and D. Fellner. Sketching subdivision surfaces. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 61–68, 2009.

[22] A. Bogush, A. Tuzikov, and S. Sheynin. 3d object reconstruction from non-parallel cross-sections. In *ICPR '04: Proceedings of the 17th International Conference on Pattern Recognition*, pages 542 – 545, 2004.

[23] J.-D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics, and Image Processing*, 44(1):1 – 29, 1988.

[24] J.-D. Boissonnat and P. Memari. Shape reconstruction from unorganized cross-sections. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 89–98, 2007.

[25] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics(Proc. SIGGRAPH 2004)*, 23(3):630–634, 2004.

[26] M. Botsch and L. Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum(Proc. Eurographics 2005)*, 24(3):611–621, 2005.

[27] M. Botsch, M. Pauly, L. Kobbelt, P. Alliez, B. Lévy, S. Bischoff, and C. Rössl. Geometric modeling based on polygonal meshes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, 2007.

[28] M. Botsch and O. Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):213–230, 2008.

[29] E. Brazil, I. Macedo, M. Sousa, L. de Figueiredo, and L. Velho. Sketching variational hermite-rbf implicits. In *SBIM '10: Proceedings of the 7th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 1–8, 2010.

[30] M. Cani, T. Igarashi, and G. Wyvill. Interactive shape design. *Synthesis Lectures on Computer Graphics and Animation*, 3(1):1–78, 2008.

[31] M. Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[32] G. Celniker and D. Gossard. Deformable curve and surface finite-elements for free-form shape design. *ACM SIGGRAPH Computer Graphics*, 25(4):257–266, 1991.

[33] X. Chen, B. Neubert, Y. Xu, O. Deussen, and S. Kang. Sketch-based tree modeling using markov random field. *ACM Transactions on Graphics(Proc. SIGGRAPH Asia 2008)*, 27(5):109:1–109:9, 2008.

[34] S.-W. Cheng and T. Dey. Improved constructions of delaunay based contour surfaces. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 322–323, 1999.

[35] J. Cherlin, F. Samavati, M. Sousa, and J. Jorge. Sketch-based modeling with few strokes. In *SCCG '05: Proceedings of the 21st Spring Conference on Computer Graphics*, pages 137–145, 2005.

[36] H. Christiansen and T. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *ACM SIGGRAPH Computer Graphics*, 12(3):187–192, 1978.

[37] J. Cohen, L. Markosian, R. Zeleznik, J. Hughes, and R. Barzel. An interface for sketching 3d curves. In *I3D '99: Proceedings of the 1999 Symposium on Interactive 3D graphics*, pages 17–21, 1999.

[38] P. Company, M. Contero, J. Conesa, and A. Piquer. An optimisation-based reconstruction engine for 3d modelling by sketching. *Computers & Graphics*, 28(6):955–979, 2004.

[39] M. Contero, F. Naya, J. Jorge, and J. Conesa. CIGRO: A minimal instruction set calligraphic interface for sketch-based modeling. In *Computational Science and Its Applications – ICCSA 2003*, 2003.

[40] F. Cordier and H. Seo. Free-form sketching of self-occluding objects. *IEEE Computer Graphics and Applications*, 27(1):50–59, 2007.

[41] B. Csébfalvi, L. Neumann, A. Kanitsar, and E. Gröller. Smooth shape-based interpolation using the conjugate gradient method. In *Proceedings of Vision, Modeling, and Visualization*, pages 123–130, 2002.

[42] C. Dance and R. Prager. Delaunay reconstruction from multiaxial planar cross-sections. Technical Report CUED/F-INFENG/TR273, Cambridge University Engineering Department, 1997.

[43] B. de Araújo and J. Jorge. Blobmaker: Free-form modelling with variational implicit surfaces. In *Proc. of the 12th Portuguese Computer Graphics Meeting*, pages 17–26, 2003.

[44] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '99: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 11–20, 1996.

[45] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics(Proc. SIGGRAPH 2003)*, 22(3):848–855, 2003.

[46] A. Dillon. *User Interface Design*. John Wiley & Sons, Inc., 2006.

[47] J. Edwards and C. Bajaj. Topologically correct reconstruction of tortuous contour forests. *Computer-Aided Design*, 43(10):1296–1306, 2011.

[48] M. Eigensatz and M. Pauly. Positional, metric, and curvature control for constraint-based surface deformation. *Computer Graphics Forum(Proc. Eurographics 2009)*, 28(2):551–558, 2009.

[49] M. Eigensatz, R. Sumner, and M. Pauly. Curvature-domain shape processing. *Computer Graphics Forum(Proc. Eurographics 2008)*, 27(2):241–250, 2008.

[50] M. Eitz, O. Sorkine, and M. Alexa. Sketch based image deformation. In *Proceedings of Vision, Modeling and Visualization (VMV) 2007*, pages 135–142, 2007.

[51] J. Fix and R. Ladner. Multiresolution banded refinement to accelerate surface reconstruction from polygons. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 240–248, 1998.

[52] T. Fleisch, G. Brunetti, P. Santos, and A. Stork. Stroke-input methods for immersive styling environments. In *SMI '04: Proceedings of the International Conference on Shape Modeling & Applications 2004*, pages 275–283, 2004.

[53] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Transactions on Graphics(Proc. SIGGRAPH 2003)*, 22(3):950–953, 2003.

[54] H. Fu, O. Au, and C. Tai. Effective derivation of similarity transformations for implicit Laplacian mesh editing. *Computer Graphics Forum*, 26(1):34–45, 2007.

[55] H. Fuchs, Z. Kedem, and S. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, 1977.

[56] J. Gain and D. Bechmann. A survey of spatial deformation from a user-centered perspective. *ACM Transactions on Graphics*, 27(4):1–21, 2008.

[57] R. Gal, O. Sorkine, N. Mitra, and D. Cohen-Or. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (Proc. SIGGRAPH 2009)*, 28(3):33:1–33:10, 2009.

[58] S. Ganapathy and T. Dennehy. A new general triangulation method for planar contours. *ACM SIGGRAPH Computer Graphics*, 16(3):69 – 75, 1982.

[59] M. Gardiner and B. Christie. *Applying cognitive psychology to user-interface design*. John Wiley & Sons, Inc., 1987.

[60] B. Geiger. Three-dimensional modeling of human organs and its application to diagnosis and surgical planning. Technical Report RR-2105, INRIA, 1993.

[61] Y. Gingold and D. Zorin. Shading-based surface editing. *ACM Transactions on Graphics(Proc. SIGGRAPH 2008)*, 27(3):1–9, 2008.

[62] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 325–334, 1999.

[63] G. Herman, J. Zheng, and C. Bucholtz. Shape-based interpolation. *IEEE Computer Graphics and Applications*, 12(3):69–79, 1992.

[64] F. Hétroy, S. Rey, C. Andújar, P. Brunet, and í. Vinacua. Mesh repair with user-friendly topology control. *Computer-Aided Design*, 43(1):101–113, 2011.

[65] K. Hildebrandt, K. Polthier, and M. Wardetzky. Smooth feature lines on surface meshes. In *SGP '05: Proceedings of the third Eurographics Symposium on Geometry processing*, 2005.

[66] D. Hoffman. *Visual Intelligence: How We Create What We See*. W. W. Norton & Company, 2000.

[67] J. Huang, X. Shi, X. Liu, K. Zhou, L. Wei, S. Teng, H. Bao, B. Guo, and H. Shum. Subspace gradient domain mesh deformation. *ACM Transactions on Graphics(Proc. SIGGRAPH 2006)*, 25(3):1126–1134, 2006.

[68] K. Hui and Y. Lai. Generating subdivision surfaces from profile curves. *Computer-Aided Design*, 39(9):783–793, 2007.

[69] T. Igarashi and J. Hughes. A suggestive interface for 3d drawing. In *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 173–181, 2001.

[70] T. Igarashi and J. Hughes. Smooth meshes for sketch-based freeform modeling. In *I3D '03: Proceedings of the 2003 Symposium on Interactive 3D graphics*, pages 139–142, 2003.

[71] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409–416, 1999.

[72] T. Ijiri, S. Owada, and T. Igarashi. The sketch l-system: Global control of tree modeling using free-form strokes. In *Smart Graphics 2006*, pages 138–146, 2006.

[73] T. Jones, F. Durand, and M. Desbrun. Non-iterative, feature-preserving mesh smoothing. In *ACM Transactions on Graphics(Proc. SIGGRAPH 2003)*, volume 22, pages 943–949, 2003.

[74] T. Ju. Fixing geometric errors on polygonal models: A survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009.

[75] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *SIGGRAPH '05: Proceedings of the 32nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 561–566, 2005.

[76] T. Ju, J. Warren, J. Carson, G. Eichele, C. Thaller, W. Chiu, M. Bello, and I. Kakadiaris. Building 3d surface networks from 2d curve networks with application to anatomical modeling. *The Visual Computer*, 21(8):764–773, 2005.

[77] T. Ju, Q.-Y. Zhou, and S.-M. Hu. Editing the topology of 3d models by sketching. *ACM Transactions on Graphics(Proc. SIGGRAPH 2007)*, 26(3):42, 2007.

[78] D. Kang, M. Masry, and H. Lipson. Reconstruction of a 3d object from a main axis system. In *AAAI Fall Symposium Series: Making Pen-Based Interaction Intelligent and Natural*, pages 92–98, 2004.

[79] L. Kara, C. D'Eramo, and K. Shimada. Pen-based styling design of 3d geometry using concept sketches and template models. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and Physical Modeling*, pages 149–160, 2006.

[80] L. Kara and K. Shimada. Sketch-based 3d-shape creation for industrial styling design. *IEEE Computer Graphics and Applications*, 27(1):60 –71, 2007.

[81] O. Karpenko and J. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Transactions on Graphics(Proc. SIGGRAPH 2006)*, 25(3):589–598, 2006.

[82] O. Karpenko, J. Hughes, and R. Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum(Proc. Eurographics 2002)*, 21(3):585–594, 2002.

[83] O. Karpenko, J. Hughes, and R. Raskar. Epipolar methods for multi-view sketching. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 167–173, 2004.

[84] N. Kehtarnavaz and R. de Figueiredo. A framework for surface reconstruction from 3d contours. *Computer Vision, Graphics, and Image Processing*, 42(1):32 – 47, 1988.

[85] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, 1975.

[86] R. Klein, A. Schilling, and W. Straßer. Reconstruction and simplification of surfaces from contours. *Graphical Models*, 62(6):429 – 443, 2000.

[87] L. Kobbelt, S. Campagna, J. Vorsatz, and H. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 105–114, 1998.

[88] L. Kobbelt, J. Vorsatz, and H. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14(1-3):5–24, 1999.

[89] D. Koel, D. Pablo, and M. Gopi. Sketching free-form surfaces using network of curves. In *SBIM '05: Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 71–78, 2005.

[90] V. Kraevoy, A. Sheffer, and M. van de Panne. Modeling from contour drawings. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 37–44, 2009.

[91] D. Laidlaw, W. Trumbore, and J. Hughes. Constructive solid geometry for polyhedral objects. *Computer Graphics (Proc. SIGGRAPH 1986)*, 20(4):161–170, 1986.

[92] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[93] S. Lee. Interactive multiresolution editing of arbitrary meshes. *Computer Graphics Forum*, 18(3):73–82, 1999.

[94] Y. Lee and F. Fang. A new hybrid method for 3d object recovery from 2d drawings and its validation against the cubic corner method and the optimisation-based method. *Computer-Aided Design*, 44(11):1090 – 1102, 2012.

[95] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo

project: 3d scanning of large statues. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, 2000.

[96] P. Liepa. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 200–205, 2003.

[97] Y. Lipman, D. Cohen-Or, R. Gal, and D. Levin. Volume and shape preservation via moving frame manipulation. *ACM Transactions on Graphics*, 26(1):5:1–5:14, 2007.

[98] Y. Lipman, D. Levin, and D. Cohen-Or. Green coordinates. *ACM Transactions on Graphics(Proc. SIGGRAPH 2008)*, 27(3):1–10, 2008.

[99] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H. Seidel. Differential coordinates for interactive mesh editing. In *SMI '04: Proceedings of the International Conference on Shape Modeling & Applications 2004*, pages 181–190, 2004.

[100] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. In *SIGGRAPH '05: Proceedings of the 32nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 479–487, 2005.

[101] L. Liu, C. Bajaj, J. Deasy, D. Low, and T. Ju. Surface reconstruction from non-parallel curve networks. *Computer Graphics Forum(Proc. Eurographics 2008)*, 27(2):155–163, 2008.

[102] Y. Liu, C. Ma, and D. Zhang. Easytoy: Plush toy design using editable sketching curves. *IEEE Computer Graphics and Applications*, 31(2):49 –57, 2011.

[103] Q. Long, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. Kang. Image-based plant modeling. *ACM Transactions on Graphics*, 25(3):599–604, 2006.

[104] K. Madsen, H. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.), 2004.

[105] J. Malik. *Interpreting line drawings of curved objects*. PhD thesis, 1986.

[106] J. Manson and S. Schaefer. Hierarchical deformation of locally rigid meshes. *Computer Graphics Forum*, 30(8):2387–2396, 2011.

[107] M. Masry and H. Lipson. A sketch-based interface for iterative design and analysis of 3d objects. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, 2007.

[108] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, pages 35–57, 2002.

[109] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228 – 258, 1992.

[110] R. Molich and J. Nielsen. Improving a human-computer dialogue. *Communications of the ACM*, 33(3):338–348, 1990.

[111] Y. Mori and T. Igarashi. Plushie: an interactive design system for plush toys. *ACM Transactions on Graphics(Proc. SIGGRAPH 2007)*, 26(3):45:1–45:8, 2007.

[112] A. Nasri. Recursive subdivision of polygonal complexes and its applications in computer-aided geometric design. *Computer Aided Geometric Design*, 17(7):595 – 619, 2000.

[113] A. Nasri, A. Abbas, and I. Hasbini. Skinning catmull-clark subdivision surfaces with incompatible cross-sectional curves. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 102–111, 2003.

[114] A. Nasri, W. Karam, and F. Samavati. Sketch-based subdivision models. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 53–60, 2009.

[115] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Transactions on Graphics(Proc. SIGGRAPH 2007)*, 26(3):41:1–41:9, 2007.

[116] A. Nealen, J. Pett, M. Alexa, and T. Igarashi. Gridmesh: Fast and high quality 2d mesh generation for interactive 3d shape modeling. In *SMI '09: Proceedings of the International Conference on Shape Modeling & Applications 2009*, pages 155–162, 2009.

[117] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics(Proc. SIGGRAPH 2005)*, 24(3):1142–1147, 2005.

[118] D. Norman. *The Design of Everyday Things*. MIT Press, 1998.

[119] Y. Ohtake, A. Belyaev, and H. Seidel. Ridge-valley lines on meshes via implicit surface fitting. In *SIGGRAPH '04: Proceedings of the 31st Annual Conference on Computer Graphics and Interactive Techniques*, pages 609–612, 2004.

[120] M. Okabe, S. Owada, and T. Igarashi. Interactive design of botanical trees using freehand sketches and example-based editing. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, 2006.

[121] J. Oliva, M. Perrin, and S. Coquillart. 3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. *Computer Graphics Forum*, 15(3):397–408, 1996.

[122] L. Olsen, F. Samavati, M. Sousa, and J. Jorge. Sketch-based mesh augmentation. In *SBIM '05: Proc. of 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2005.

[123] L. Olsen, F. Samavati, M. Sousa, and J. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, 2009.

[124] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, 2008. Available from: <http://www.openmp.org/mp-documents/spec30.pdf>.

[125] G. Orbay and L. Kara. Sketch-based modeling of smooth surfaces using adaptive curve networks. In *SBIM '11: Proceedings of the 8th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 71–78, 2011.

[126] B. Payne and A. Toga. Surface reconstruction by multiaxial triangulation. *IEEE Computer Graphics and Applications*, 14(6):28 –35, 1994.

[127] Q. Peng, X. Jin, and J. Feng. Arc-length-based axial deformation and length preserved animation. In *CA '97: Proceedings of the Computer Animation*, pages 86–92, 1997.

[128] D. Perkins. Cubic corners. quarterly progress report. *MIT Research Laboratory of Electronics*, pages 207–214, 1968.

[129] T. Popa, D. Julius, and A. Sheffer. Material aware mesh deformations. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*, pages 5:1–5:2, 2005.

[130] D. Pugh. Designing solid objects using interactive sketch interpretation. In *I3D '92: Proceedings of the 1992 Symposium on Interactive 3D graphics*, pages 117–126, 1992.

[131] S. Raya and J. Udupa. Shape-based interpolation of multidimensional objects. *IEEE Transactions on Medical Imaging*, 9(1):32 –42, 1990.

[132] A. Rivers, F. Durand, and T. Igarashi. 3d modeling with silhouettes. *ACM Transactions on Graphics(Proc. SIGGRAPH 2010)*, 29(4):109:1–109:8, 2010.

[133] K. Rose, A. Sheffer, J. Wither, M. Cani, and B. Thibert. Developable surfaces from arbitrary sketched boundaries. In *SGP '07: Proceedings of the fifth Eurographics Symposium on Geometry Processing*, pages 163–172, 2007.

[134] S. Schaefer, J. Warren, and D. Zorin. Lofting curve networks using subdivision surfaces. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 103–114, 2004.

[135] R. Schmidt and K. Singh. Sketch-based procedural surface modeling and compositing using surface trees. *Computer Graphics Forum(Proc. Eurographics 2008)*, 27(2):321–330, 2008.

[136] R. Schmidt, K. Singh, and R. Balakrishnan. Sketching and composing widgets for 3d manipulation. *Computer Graphics Forum(Proc. Eurographics 2008)*, 27(2):301–310, 2008.

[137] R. Schmidt, B. Wyvill, M. Sousa, and J. Jorge. Shapeshop: sketch-based solid modeling with blobtrees. In *SBIM '05: Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–62, 2005.

[138] V. Schomaker, J. Waser, R. Marsh, and G. Bergman. To fit a plane or a line to a set of points by least squares. *Acta Crystallographica*, 12(8):600–604, 1959.

[139] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 20(4):151–160, 1986.

[140] A. Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008.

[141] A. Sheffer and V. Kraevoy. Pyramid coordinates for morphing and deformation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 68–75, 2004.

[142] K. Shoemake. Arcball: a user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the conference on Graphics Interface 1992*, pages 151–156, 1992.

[143] K. Singh and E. Fiume. Wires: a geometric deformation technique. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 405–414, 1998.

[144] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *SGP '07: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 109–116, 2007.

[145] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H. Seidel. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188, 2004.

[146] R. Sowell, L. Liu, T. Ju, C. Grimm, C. Abraham, G. Gokhroo, and D. Low. Volume viewer: an interactive tool for fitting surfaces to volume data. In *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 141–148, 2009.

[147] M. Sugihara, B. Wyvill, and R. Schmidt. WarpCurves: A tool for explicit manipulation of implicit surfaces. *Computers & Graphics*, 34(3):282–291, 2010.

[148] C. Tai, H. Zhang, and J. Fong. Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum*, 23(1):71–83, 2004.

[149] S. Tano, T. Kodera, T. Nakashima, I. Kawano, K. Nakanishi, G. Hamagishi, M. Inoue, A. Watanabe, T. Okamoto, K. Kawagoe, K. Kaneko, T. Hotta, and M. Tatsuoka. Godzilla: Seamless 2d and 3d sketch environment for reflective and creative design work. In *Human-Computer Interaction (INTERACT'03)*, pages 311–318, 2003.

[150] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 205–214, 1987.

[151] G. Turk and J. O'Brien. Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 335–342, 1999.

[152] E. Turquin, J. Wither, L. Boissieux, M. Cani, and J. Hughes. A sketch-based interface for clothing virtual characters. *IEEE Computer Graphics and Applications*, 27(1):72–81, 2007.

[153] P. Varley, R. Martin, and H. Suzuki. Frontal geometry from sketches of engineering objects: is line labelling necessary? *Computer-Aided Design*, 37(12):1285–1307, 2005.

[154] P. Varley, Y. Takahashi, J. Mitani, and H. Suzuki. A two-stage approach for interpreting line drawings of curved objects. In *SBIM '04: Proceedings of the 1st Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 117–126, 2004.

[155] H. Wang and L. Markosian. Free-form sketch. In *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–58, 2007.

[156] Y. Wang and J. Aggarwal. Surface reconstruction and representation of 3-d scenes. *Pattern Recognition*, 19(3):197 – 207, 1986.

[157] D. Weinstein. Scanline surfacing: building separating surfaces from planar contours. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 283–289, 2000.

[158] W. Welch and A. Witkin. Variational surface modeling. In *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 157–166, 1992.

[159] P. Werahera, G. Miller, G. Taylor, T. Brubaker, F. Daneshgari, and E. Crawford. A 3-d reconstruction algorithm for interpolation and extrapolation of planar cross sectional data. *IEEE Transactions on Medical Imaging*, 14(4):765 –771, 1995.

[160] L. Williams and A. Hanson. Perceptual completion of occluded surfaces. *Computer Vision and Image Understanding*, 64(1):1–20, 1996.

[161] J. Wither, F. Bertails, and M. Cani. Realistic hair from a sketch. In *SMI '07: Proceedings of the Shape Modeling International & Applications 2007*, pages 33–42, 2007.

[162] J. Wither, A. Bouthors, and M. Cani. Rapid sketch modeling of clouds. In *SBIM '08: Proceedings of the 5th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2008.

[163] W. Xu and K. Zhou. Gradient domain mesh deformation ł a survey. *Journal of Computer Science and Technology*, 24(1):6–18, 2009.

[164] W. Xu, K. Zhou, Y. Yu, Q. Tan, Q. Peng, and B. Guo. Gradient domain editing of deforming mesh sequences. *ACM Transactions on Graphics(Proc. SIGGRAPH 2007)*, 26(3), 2007.

[165] S. Yoshizawa and A. Belyaev. Fair triangle mesh generation with discrete elastica. In *GMP '02: Proceedings of the Geometric Modeling and Processing; Theory and Applications*, page 119, 2002.

[166] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H. Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: Proceedings of the 31st Annual Conference on Computer Graphics and Interactive Techniques*, pages 644–651, 2004.

[167] R. Zayer, C. Rössl, Z. Karni, and H. Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum(Proc. Eurographics 2005)*, 24(3):601–609, 2005.

[168] L. Zhang, N. Snavely, B. Curless, and S. Seitz. Spacetime faces: high resolution capture for modeling and animation. *ACM Transactions on Graphics(Proc. SIGGRAPH 2004)*, 23(3):548–558, 2004.

[169] X. Zhang, W. Chen, J. Fang, R. Wang, and Q. Peng. Perceptually-motivated shape exaggeration. *The Visual Computer*, 26(6):985–995, 2010.

[170] Y. Zheng, H. Fu, D. Cohen-Or, O. Au, and C. Tai. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum (Proc. Eurographics 2010)*, 30(2):563–572, 2011.

[171] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H. Shum. Large mesh deformation using the volumetric graph Laplacian. *ACM Transactions on Graphics(Proc. SIGGRAPH 2005)*, 24(3):496–503, 2005.

[172] Q. Zhou, T. Weinkauf, and O. Sorkine. Feature-based mesh editing. In *Proc. Eurographics 2011, Short Papers*, pages 1–4, 2011.

[173] J. Zimmermann, A. Nealen, and M. Alexa. Silsketch: automated sketch-based editing of surface meshes. In *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 23–30, 2007.

[174] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 259–268, 1997.

Author's Publications

[1] Kai Wang, Jianmin Zheng, and Hock-Soon Seah, "Topology-editable surface reconstruction from cross sections with arbitrary orientation," submitted to Computer-Aided Design, 2013.

[2] Kai Wang, Jianmin Zheng, and Hock-Soon Seah, "Enhancing sketching and sculpting for sketch-based modeling," submitted to Computer & Graphics, 2013.

[3] Jianjiang Pan, Jianmin Zheng, and Kai Wang, "Semi-reversible watermarking of 3d triangular mesh models based on improved prediction-error expansion," submitted to IEEE Transactions on Visualization and Computer Graphics, 2013.

[4] Kai Wang, Jianmin Zheng, Hock-Soon Seah, and Yuewen Ma, "Triangular mesh deformation via edge-based graph," *Computer-Aided Design and Applications* (**Best Paper Award** from 2011 International Conference on Computer-Aided Design and Exhibition), Vol.9, No.3, 2012, pp.345-359.

[5] Yuewen Ma, Jianmin Zheng, and Kai Wang, "Physically-based NURBS surface editing with curves," *Computer-Aided Design and Applications*, Vol.9, No.3, 2012, pp.361-374.

[6] Kai Wang, Jianmin Zheng, and Hock-Soon Seah, "Reference plane assisted sketching interface for 3D freeform shape design," in *CYBERWORLDS 2010: Proceedings of the 2010 International Conference on Cyberworlds*, IEEE, 2010, pp.105-112.

[7] Kai Wang, Jianmin Zheng, and Hock-Soon Seah, “Edge-based Laplacian mesh deformation,” in *NICOGRAPH 2010: Proceedings of NICOGRAPH International 2010*, 2010, pp.1-6.

[8] Kai Wang, Jianmin Zheng, and Hock-Soon Seah, “Flexible surface deformation for mesh editing,” in *IWAIT 2010: Proceedings of the International Workshop on Advanced Image Technology 2010*, 2010, pp.13-18.