3D Object Reconstruction from 2D and 3D Line Drawings

CHEN, Yu

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Information Engineering

June, 2008

Abstract

3D object design from line drawings, either in 2D or 3D, is an important problem in both computer vision and graphics. It has a wide range of applications including flexible 3D sketch input in CAD, computer game, webpage content design, image based object modeling, and 3D object retrieval, etc.

Currently, a large number of 3D object design tools focus on reconstructing 3D objects from sketches on the 2D plane in the form of line drawings. A line drawing is defined as the 2D projection of the edges and vertices of a 3D object in a generic view. Many methods have been put forward to solve this problem, but they usually fail when the geometric structure of a 3D object becomes complex. In the first part of this thesis, a novel approach based on a divide-and-conquer strategy is proposed to handle 3D reconstruction of complex manifold objects from single 2D line drawings. The approach consists of four steps: 1) identifying the real faces and internal faces from a line drawing; 2) separating the line drawing into multiple simpler line drawings based on the result of face identification; 3) reconstructing the 3D shapes from these simpler line drawings; and 4) merging the 3D shapes into one complete object represented by the original line drawing. A number of examples are given to show that our approach can handle 3D reconstruction of more complex objects than previous methods.

On the other hand, we also delve into the problem of designing 3D objects by reconstructing them from ling drawings sketched in 3D space. In order to let the user design directly in 3D space, virtual 3D environments are often used. Unfortunately, these virtual-reality-based methods have the drawbacks that there are awkward

devices worn by the user and the virtual environment systems are expensive. In the second part of this thesis, we propose a novel vision-based approach to 3D object design, which leads to an inexpensive and convenient way for sketching in 3D space. Our system consists of a PC, a camera, and a mirror, which are inexpensive and easy to set up. We use the camera and mirror to track a wand so that the user can design 3D objects by sketching in 3D free space directly without having to wear any cumbersome devices. In addition, a number of sketching and editing operations are developed to facilitate object design. The system provides designers a whole new user interface of designing 3D objects conveniently.

摘要

從二維或三維線條圖中重建三維物體是計算機視覺和圖形學的重要研究問題之一。這一研究在計算機輔助設計（CAD）中的三維草圖輸入，計算機遊戲，網頁設計，基於網頁的物體建模和三維物體獲取等方面具有廣泛的應用。

目前，許多三維物體設計工具都是基於從二維平面草圖中重建三維物體的方法，而二維草圖通常可以線條圖的方式給出。線條圖是三維物體的頂點和邊在二維平面上的投影。在此之前，已經有不少研究工作試圖解決從線條圖重建三維物體這一問題，並提出了很多方法，但這些方法往往無法處理具有復雜幾何結構的三維物體。在本論文的前半部分中，我們提出一種基於“分而治之”思路，專門針對從單個二維線條圖中重建複雜三維流形物體的方法。該方法主要由如下四個步驟構成：（1）根據線條圖檢測三維物體的內外表面；（2）利用線條圖的面檢測的結果，將一個複雜的線條圖拆解成若干個簡單的線條圖；（3）從得到的簡單線條圖中重建相應的三維物體；（4）將得到的三維物體合併為一個複雜物體，對應原先的複雜線條圖。大量的實驗結果證明我們的方法相比過去的方法可以用於重構更為複雜的三維物體。

在另一方面，我們同時深入研究了通過在三維空間中作草圖的方法直接設計三維物體的設計方法。通常，為了使用戶能夠直接在三維空間中作畫，往往需要構建三維的虛擬現實環境。這些基於虛擬現實的設計方法的主要缺點在於用戶往往被要求佩戴特殊且笨拙的裝備，而那些虛擬現實的系統往往又非常昂貴。在本論文的後半部分中，我們提出一種新的基於視覺的三維物體設計方法。通過使用這一方法，三維設計可以變得廉價而方便。我們的系統由個人計算機，一個攝像頭，及一面鏡子組成，這些設備並不昂貴且易於設置。我們利用攝像頭和鏡子跟踪用戶手上的畫棒，因此用戶可以直接在三維空間中作圖而不需要佩戴任何笨拙的裝備。除此之

外，我們還在系統中開發了一系列用於簡化三維設計的編輯和操作方式。這一新系統為設計者提供了一種設計三維物體的全新用戶界面。

Acknowledgement

First and foremost, I would like to give my sincere gratitude to my supervisor Prof. Jianzhuang Liu, who was guiding and supporting me during the past two years. I was my great honor to work under the guidance of Prof. Liu, a nice person and a great researcher. As a mentor, he is always glad to talk with students and always willing to teach something to his students. His valuable comments, constructive suggestions, and close guidance are of great help not only in the thesis writing but also in the way the research is carried out.

I am also grateful to my co-supervisor Prof. Xiaoou Tang. His intuition and deep insights provide me a never ending stream of ideas, suggestions and comments. It is Prof. Tang's encouragement and direction that makes me be proud of myself in research and not lose the impetus of continuous working.

Also, many thanks to all the members of the Multimedia Lab, Shifeng Chen, Zhenguo Li, Wei Liu, Pengfei Shan, Huan Wang, Chunjing Xu, Wei Zhang, etc. My research owes much to the inspiring discussion with them and their consistent technical assistance both in programming and paper writing during the past two years. I am very proud of this experience of working in such a cohesive and productive group.

Finally and the most important, I would like to thank my parents for their constant support and encouragement whenever I meet difficulties, which keeps me in high spirit in my research.

Contents

Chapter 1

Introduction and Related Work

Historically, 3D object design, usually related to architectural designs and sculptures, was treated as a mean of art rather than a technology until the advent of information age. Now, with the advancement in computer technology, 3D object design has found its wide application in computer game, webpage content design, object modeling, etc. As a consequence of commercial and technological impetus, numerous researches in computer vision and graphics have been conducted to facilitate the designing process by introducing new modeling schemes and improving the user interfaces over the last several decades.

This thesis focuses on 3D object design from line drawings and sketching. The sketching input can be given in either 2D plane or 3D space, and usually they result in quite different methods and techniques for reconstructing 3D objects. Since both types of inputs are considered in our work, the thesis actually addresses following two different issues in 3D object design.

First, when 2D inputs (usually in the form of line drawings) are given, the problem of 3D reconstruction based on 2D line drawings is investigated, which is an important computer vision problem and a branch of 3D object design. This research initiates from the fact that traditional architectural engineers use manual sketching tools such as paper and pencil to rough out their initial ideas. The goal of this research is to automatically recover the depth information and infer the 3D geometrical and topological structure of the objects represented by input line

drawings in 2D sketching plane. Human has no difficulty in interpreting these line drawings, but the same goal can be challenging for the computer due to ambiguity from the lost dimension. The first part of this thesis is in the pursuit of this goal.

Second, when sketches are given in 3D space, we actually turn to the problem of direct 3D modeling, that is, to construct 3D objects directly in the 3D space. Such a direct 3D design is traditionally completed in a special virtual 3D environment. This thesis presents a complete novel framework for direct 3D design by means of sketching in the air, which overcomes several drawbacks of virtual-reality-based methods.

The rest of this chapter is organized as follows. A detailed introduction to the problem of 3D reconstruction from 2D line drawings and its related work are presented in Sections 1.1, 1.2, and 1.3. A general review to current 3D modeling systems is given in Section 1.4. Finally, Section 1.5 summarizes the two main contributions of this thesis to the problem of sketch-based 3D object design.

1.1 Reconstruction from 2D Line Drawings and the Applications

Currently, a large number of 3D object design tools focus on reconstructing 3D objects from line drawings on the 2D plane. A line drawing is defined as the 2D projection of the edges and vertices of a 3D object in a generic view, with or without hidden lines invisible. It is the simplest and most straightforward way of illustrating a 3D object. The human vision system has the ability to interpret 2D line drawings as 3D objects without difficulty. Emulating this ability is an interesting research topic for enhancing machine vision system. It is highly desirable to develop algorithms that can understand the single 2D line drawing and reconstruct 3D geometry of the object it represents. Fig. 1.1 shows an example of 3D reconstruction from line drawings.

These line drawings can be generated by sketching on the screen with a mouse

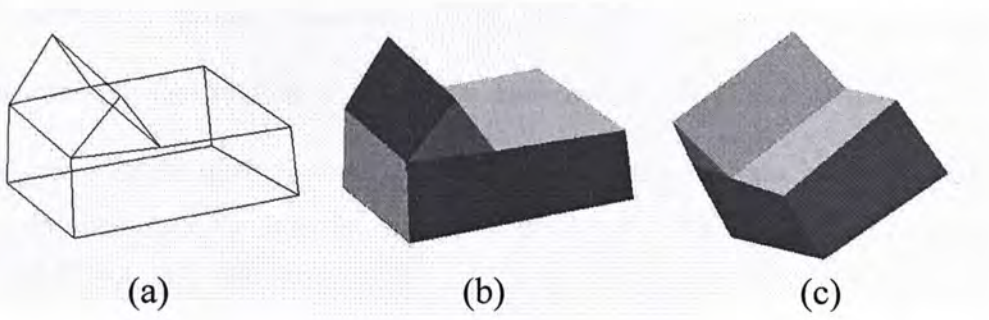


Fig. 1.1: An example of 3D reconstruction from line drawings. (a) A 2D line drawing. (b) Reconstructed 3D object. (c) Reconstructed 3D object in another view.

or a tablet PC pen and on paper with a pen. They can be represented by single edge-vertex graphs. In the case that a line drawing obtained by scanning the sketch image on paper, extracting an edge-vertex graph require some pre-processing such as binarizing and thinning of the image, and tracking and analyzing of the lines and vertices. Compared with line drawings without hidden lines, a line drawing with hidden lines shown makes it possible to reconstruct a complete and more complex object, including its back, from the line drawing. Much work concerning line drawings has been published in computer vision literature [9], [10], [11], [13], [43], [51], [52], [53], [50], [58], [65], [71], [72], [76], [77], and in CAD and graphics [3], [8], [16], [17], [24], [49], [59], [61], [81], [84], [86], [85].

With the development of computer vision research and the increasing requirement of interaction between human and computer, 3D reconstruction from line drawings find more applications, such as

- providing a flexible sketching input for conceptual designers who tend to prefer pencil and paper to mouse and keyboard in current CAD systems [8], [49], [68], [72];
- automatic conversion of existing industrial wireframe models to solid models [3], [8];
- building rich databases for object recognition systems and reverse engineering

algorithms for shape reasoning [3], [8], [17], [78];

- interactive generation of 3D models from images [24], [73], [81];
- user-friendly query interface for 3D object retrieval from large 3D object databases and the internet, which lets users easily draw 3D objects as queries [11], [59], [61], [66], [92].

1.2 Previous Work on 3D Reconstruction from Single 2D Line Drawings

To reconstruct the 3D object from single 2D line drawing, the main stream approach in the previous researches is to formulate the problem as an optimization problem based on different objective functions.

Marill [58] presented his method based on a simple criterion: minimizing the standard deviation of the angles in the reconstructed object, which is called the MSDA principle. This criterion can be used to inflate a 2D line drawing into a 3D shape. Marill's approach is tolerant of freehand sketching errors, but cannot reconstruct complete 3D objects if their hidden lines are not drawn. Motivated by the MSDA, Brown and Wang [9] proposed to minimize the standard deviation of the segment magnitudes (MSDSM) in the recovered planar object. More recently, Shoji et al. [71] presented the criterion of minimizing the entropy of angle distribution (MEAD), and claimed that it is more general than both the MSDA and the MSDSM.

MSDA, MSDSM, and MEAD are criteria of regularity that humans perceive when interpreting 2D line drawings. Later researches [43], [49], [64], [68], [87] follow this idea and incorporate more heuristic regularities in the reconstruction. Leclerc and Fischler's approach [43] considers not only the MSDA principle, but also the planarity constraint exhibited on the faces of a planar object. The method in [64] and [87] concentrates on the reconstruction of symmetric polyhedra by developing a regularity of model symmetry. Lipson and Shpitalni [49] took Leclerc and Fischler's

work further by using more constraints for the reconstruction, such as line parallelism, isometry, corner orthogonality, and skewed face orthogonality, which are in accordance with human visual perception of line drawings. All these constraints are combined together to form an objective function. Lipson and Shpitalni's method [49] can handle more complex objects than all the previous methods. Later, Shesh and Chen applied Lipson and Shpitalni's algorithm to their sketching system [68]. In [50], Liu et al. use the parameters of the planes that pass through the planar faces of an object to be the variables of the objective function. Their method is more robust and computationally efficient than the previous methods.

These optimization-based methods above can handle only planar objects and require all the hidden lines to be drawn, and the variables of the objective functions are usually the lost depths of the vertices in the line drawing. Recently, some attempts [10], [12], [83] have been made to recover a complete solid from a line drawing with visible lines only (without hidden lines), but these methods are only applicable to relatively simple objects and require a lot of human interactions. The methods in [11] and [86] try to handle curved-faced object reconstruction. The main limitation of them is that they need human interaction. Besides, the former can only deal with simple objects and the latter is limited to symmetric objects without holes. Also, the shading information is used in [69] and [70] to recover the visible surfaces of 3D polyhedra in images from the edges of the polyhedra.

1.3 Other Related Work on Interpretation of 2D Line Drawings

There is much work done on the other areas of line drawings interpretation besides 3D reconstruction from single line drawings in the literature. Although these researches may not focus on a direct 3D reconstruction as those mentioned in the last section, they are also useful and related to our work. In this section, we give a brief review on these related papers.

1.3.1 Line Labeling and Superstrictness Problem

Since the early stage of computer vision, a large amount of the work is about line labeling and 3D reconstruction based on a labeled drawing [15], [18], [19], [20], [21], [30], [32], [35], [57], [76], [75], [89]. Line labeling aims to find a set of consistent labels from a line drawing, and it does not explicitly give the 3D structure represented by a line drawing. Early work on line labeling focuses on labeling polyhedra without hidden lines. Recently, Cooper extends this line labeling research to wireframes with hidden lines visible as well as curved objects [19], [20], [21]. One limitation of line labeling is that multiple consistent labeling solutions for one line drawing are possible [69].

Another body of work on line drawing interpretation is related to judging the correctness of line drawings and give their possible reconstruction based on algebra test with linear equalities and inequalities [55], [65], [74], [76], [75], [82]. The problem of these methods is that such a formulation is superstrict and not robust; an originally correct line drawing will be judged as *impossible* after a little deviation of one or more vertices, causing a 3D reconstruction to fail [65].

1.3.2 CAD Reconstruction

This group of work focuses on reconstructing a 3D CAD model from its multiple (usually three) orthographic projections [2], [39], [42], [45], [54]. Since more information is available from three orthogonal views for the reconstruction task, such work is much easier than the reconstruction from one projection.

1.3.3 Modeling from Images

Tour into the image (TIP) proposed by Horry et al. [31] and later improved by Kang et al. [36] construct simple 3D scene models using vanishing points and a spidery mesh. Their researches are dependent on the directly designation of vanishing points and spidery meshes by the user. TIP can achieve impressive results and is easy to use. However, it is not strict in geometry and limited to the implied situation where

the image plane must be vertical to the ground plane.

More strictly reconstruction work is on the basis of single view metrology (SVM) [48], [40], [73]. Their methods found on calibration of camera intrinsic parameters, thus do not restrict to the assumption that the image plane is vertical to the image plane. The main disadvantage of these methods, however, is that they require a lot of human interaction. Such system requires the user not only to set enough constraints to decide the camera intrinsic parameter, but also to give hints to every plane or vertex. For example, to define a point, the user must tell the vertical projection of that point on to the ground plane or another reference plane [48], [40]. Also, to define a plane, the user should tell its vanishing line. Such user input schemes are burdensome since they are not much easier than inputting an object using a traditional 3D modeler such as AutoCAD [34].

Our work is different from these researches in the sense that we requires no human interactions and our reconstruction methods process single line drawings instead of images.

1.3.4 Identifying Faces in the Line Drawings

Face identification from a line drawing provides necessary information for reconstructing 3D objects. An object consists of faces. If the face configuration of an object is known before the reconstruction of its 3D geometry, the complexity of the reconstruction will be reduced significantly. Fig. 1.2(b) and (d) show the faces of the two line drawings in Figs. 1.2(a) and (c).

In general, there are many cycles in a line drawing and only a small subset of them represents its faces, and the number of cycles grows exponentially with the number of edges. Thus finding the faces from a line drawing is not a trivial problem.

Much effort has been made in this area for the past two decades [3], [8], [23], [43], [46], [51], [52], [53], [68], [72]. Among these techniques, the algorithms presented in [52] and [53] are useful for our work.

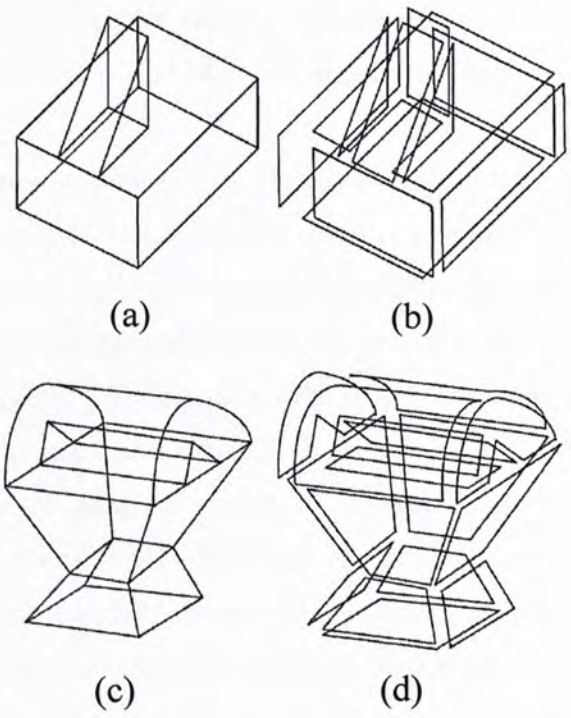


Fig. 1.2: The faces of the two line drawings.

1.4 3D Modeling Systems

Great effort has been made to develop CAD systems for 3D model design in the past three decades. Current techniques can be classified into the following four categories:

1) Traditional CAD tools, such as AutoCAD [34] and SolidWorks [22]. These tools are sophisticated systems suitable for engineers to input precise geometry of models. However, they are not suitable for designers to rapidly express their ideas at the initial stage of model development.

2) Automatic 3D object reconstruction from 2D line drawings. This is one of the main research topics in computer vision and graphics. The methods are mainly based on line labeling, algebra, image regularities, and optimization [19], [43], [49], [58], [68], [69], [76], [77], [81], [82], [90]. A complete review to these methods and related systems has already been given in Section 1.2 and Section 1.3. Compared

with other 3D modeling systems, systems based on reconstruction are automatic or semi-automatic and they require much fewer interaction from users. But the critical problem in these systems is that they can handle only relatively simple objects at the current stage.

3) 2D-sketch-based modeling user interfaces. Most traditional designers still prefer pencil and paper to mouse and keyboard in current CAD systems to sketch their ideas of shapes [1]. To bridge the gap between the flexible 2D sketches and the rigid CAD systems, researchers have developed tools that try to convert 2D sketches into 3D models [4], [26], [33], [37], [60], [62], [68], [93], giving rise to the state-of-the-art systems including Quick-sketch [26], SKETCH [93], Teddy [33], FiberMesh [60], and etc. A more complete survey on 2D-sketch-based modeling techniques and systems can be found in [17]. However, one physical limitation that cannot be overcome by these tools is that the sketching and editing operations are performed on a 2D plane (tablet or screen). With one dimension missing, the 3D positions of the strokes, surfaces, and objects drawn on a 2D plane are often ambiguous.

4) Design in virtual 3D environments. More than ten years ago when virtual reality (VR) techniques appeared, experts already noticed that VR was a perfect CAD system and predicted that it would come to replace the ordinary CAD because the designer could work naturally and intuitively in a real 3D environment [63]. However, until now this prediction has not come true due to the cost of the equipments, the inflexibility to use, and the slow frame update rates. Researchers are trying to develop better techniques for 3D design in VR [5], [25], [29], [38], [56], [67] but they need special and awkward devices to operate by, or connect to, the user, making the design an unnatural process.

From the discussion above, we can see that the current methods for 3D model design are not good enough. Researchers still need to develop more friendly and inexpensive interfaces with better design methodology.

1.5 Research Problems and Our Contributions

There have been a number of papers discussing the 3D reconstruction from single 2D line drawings [9], [10], [11], [43], [49], [58], [71], [86] and 3D modeling interfaces [4], [5], [26], [29], [33], [37], [38], [56], [62], [68], [67], [93]. In previous sections, we have reviewed the pros and cons of these previous researches. This thesis mainly concentrates on the following two topics in view of the limitations existing in the previous work:

First, previous methods for line drawing reconstruction usually fail for complex objects, which may contain internal faces or holes. In light of this problem, we present a novel divide-and-conquer approach which is mainly devoted to handle those complex objects. To the best knowledge of us, our approach can tackle the objects with the most complex geometrical structure among all the researches in this area.

Second, considering the advantages and limitations of all the four categories of the 3D modeling systems discussed in Section 1.4, we propose a novel vision-based approach to 3D object design, which leads to an inexpensive and convenient way of designing 3D objects by directly reconstructing them from sketches in 3D space.

1.5.1 Recovering Complex Manifold Objects from Line Drawings

Many methods have been put forward to solve the problem of 3D object reconstruction from a single 2D line drawing, but most of these researches concerning the 3D reconstruction from single 2D line drawings consider only relatively simple objects. They usually fail when the geometric structure of a 3D object becomes complex, e.g., objects which contains internal faces or holes.

Chapter 2 proposes a novel divide-and-conquer approach to 3D reconstruction of complex manifold objects from single 2D line drawings, as shown in the following figure. Theoretically, we prove that the partition of a line drawing into simpler line drawings along its internal faces exists and is unique. Also, our work is the first attempt to investigate the internal faces and perform separation on the line drawing

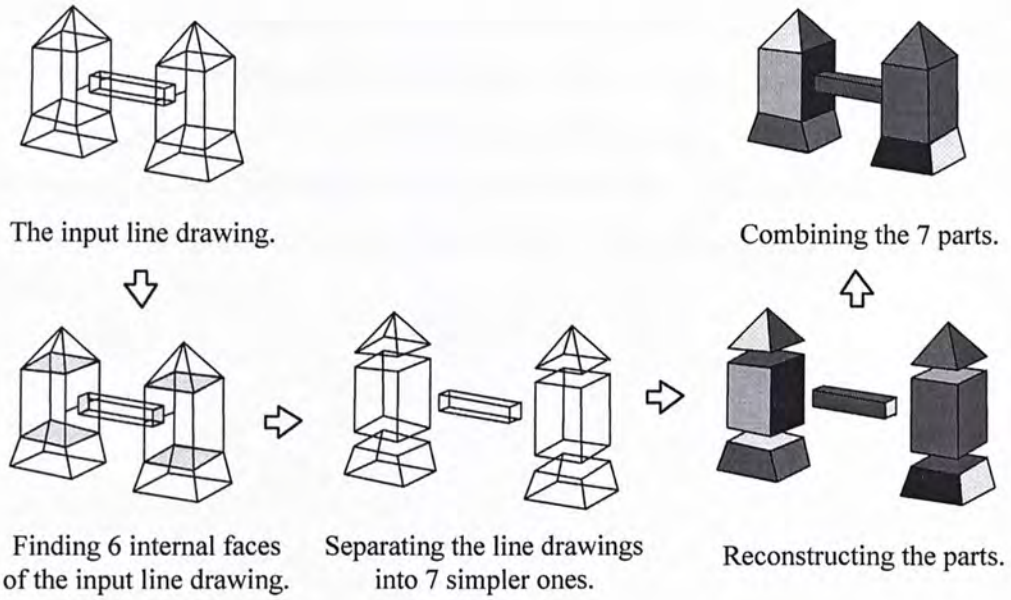


Fig. 1.3: Illustration of our divide-and-conquer approach to the problem of line drawing reconstruction.

automatically.

We use those high complex line drawings from the past papers and also construct those which never appear in the previous work to verify the effectiveness our new approach, and the experiments show that our approach can tackle 3D reconstruction from much more complex line drawings than previous algorithms. Fig. 1.3 illustrates the process of our approach. This work is published in ICCV 2007[13].

1.5.2 The Vision-based Sketching System

Most current 3D object design tools work on a 2D drawing plane such as computer screen or tablet, which is often inflexible with one dimension lost. On the other hand, virtual reality based methods have the drawbacks that there are awkward devices worn by the user and the virtual environment systems are expensive.

In Chapter 3, we propose a novel vision-based approach to 3D object design. Our system consists of a PC, a camera, and a mirror. We use the camera and mirror to track a wand so that the user can design 3D objects by sketching in 3D

free space directly without having to wear any cumbersome devices. A number of new techniques are developed for working in this system, including input of object wireframes, gestures for editing and drawing objects, and optimization-based planar and curved surface generation. Our system provides designers a whole new user interface for designing 3D objects conveniently. This system is published in CVPR 2008 [14].

Chapter 2

Reconstruction from Complex Line Drawings

2.1 Introduction

A line drawing is the 2D projection of the wireframe of an object. Humans have no difficulty in perceiving the 3D geometry from a 2D line drawing. Emulating this ability is an important research topic in both computer vision and graphics. Much work has been carried out on 3D reconstruction from line drawings in the past three decades. However, when a line drawing becomes complex, previous methods usually fail to obtain a desired object due to two reasons: 1) the methods are not powerful enough to tackle the 3D reconstruction of complex objects; and/or 2) the algorithms can easily get trapped into local optima.

In this chapter, we propose a novel divide-and-conquer approach to the 3D reconstruction of complex planar-faced manifold objects from single 2D line drawings with hidden lines visible. Manifolds belong to a class of most common solids, the definition of which is given in Section 2.3. Our approach is based on the fact that a complex object is in general the combination of less complex objects, each of which is easier to reconstruct. Fig. 2.1 shows an example where a line drawing is decomposed into three simpler ones. Obviously, the 3D reconstruction from each of the three is an easier job than the reconstruction from the original line drawing. Our approach

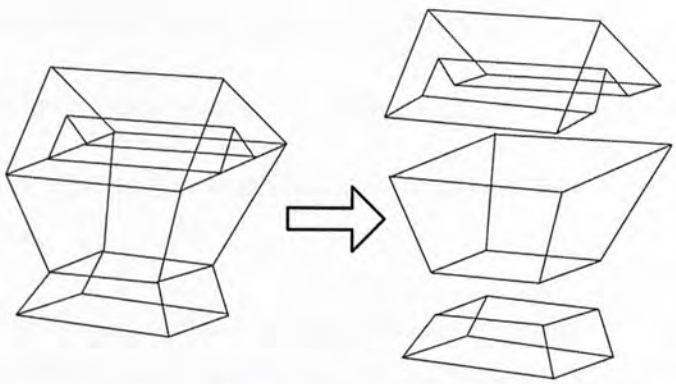


Fig. 2.1: Separating a line drawing into three simpler ones.

is summarized as four steps: 1) identifying the internal faces from the line drawing, 2) separating the line drawing into less complex ones based on the result of face identification from the complex line drawing, 3) reconstructing the 3D shape from each of these simpler line drawings, and 4) merging these 3D shapes into a complete object. Line drawings we discuss in this chapter are with hidden lines visible.

Among the related previous work, the state-of-the-art method by Lipson and Shpitalni [49] is effective for reconstructing planar objects. It uses thirteen criteria for the reconstruction, such as MSDA, face planarity, and line parallelism. From our experiments, we found that their algorithm fails to obtain an expected 3D object from a line drawing when the geometry of the object becomes more complex. In fact, we can see this problem in the previous methods from the relatively simple reconstructed 3D objects shown in the previous papers.

The rest of this chapter is organized as following. Section 2.2 states the assumptions for the reconstruction problem and defines terms that are frequently used in the chapter. In Section 2.3, we propose our method for the separation of a complex line drawing into simpler ones. Section 2.4 presents the reconstruction algorithm for merging the 3D objects that are recovered from the simpler line drawings. A number of experimental results are shown in Section 2.5 and finally Section 2.6 summarizes the chapter.

2.2 Assumptions and Terminology

The following assumptions are made before we formulate the reconstruction problem.

Assumption 2.1 The object represented by a line drawing is a manifold whose faces are all planar.

Assumption 2.2 A line drawing is the parallel or near-parallel projection of a wireframe manifold in a generic view where all the edges and vertices of the manifold are visible, and it can be represented by a single edge-vertex graph[1] .

Assumption 2.3 All the real faces of the manifold a line drawing represents have been available.

Assumption 2.4 All internal faces are planar.

So far there has been little work on automatic 3D reconstruction from single 2D line drawings representing objects with curved faces. In this chapter, we focus on a class of most common solids, called manifolds, with planar faces. A line drawing is said to be the projection of a wireframe in a *generic view* means that the topology of the line drawing is preserved under slightly variations of the viewpoint. In this case, no two vertices appear at the same position, no two edges overlap in the 2D projection plane, and 3D non-collinear edges are not projected as collinear edges. Face identification from line drawings with hidden lines visible has been studied extensively [3], [8], [43], [47], [51], [52], [53], [72], and the algorithms developed in [47] and [52] can be used to find the faces from a line drawing. Therefore, we have Assumption 2.3 in this chapter. The reason to have Assumption 2.4 is stated at the end of this section after the definition of an internal face is given.

For better understanding the content in the following sections, we here summarize the terms that appear in the rest of the chapter.

[1]The crossing point of two lines is not a vertex.

- **Manifold**. A manifold, or more rigorously 2-manifold, is a solid where every point on its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space [6]. In this chapter, we consider such manifolds that are made up by flat surfaces. A basic property of a manifold is that each edge is shared exactly by two faces [41].

- **Face (real face)**. A face is one of the flat surfaces that make up a manifold. In what follows, We call it a *real face* to distinguish it from an internal face defined below.

- **Internal face**. An internal face is a face inside a manifold only with its edges visible on the surface. It is not a real face but is formed by gluing two manifolds together.

- **Edge**. An edge of a line drawing is the intersection of two non-coplanar real faces. An edge e is also denoted by $\{v_{e1}, v_{e2}\}$ where v_{e1} and v_{e2} are two vertices of e.

- **Artificial line**. An artificial line is a line used to indicate the coplanar relationship of two cycles. It is generated by the designer of the sketch.

- **Cycle**. A cycle is formed by a sequence of vertices $v_0, v_1, \cdots, v_n$, where $n \geq 3$, $v_0 = v_n$, the n vertices are distinct, and there exists an edge connecting v_i and v_{i+1} for $i = 0, 1, \cdots, n-1$. A cycle is denoted by $(v_0, v_1, \cdots, v_n)$. Since the boundary of a face is a cycle, a face is denoted the same way as a cycle.

- **Chord**. A chord of a cycle is an edge that connects two nonadjacent vertices of the cycle.

- **Vertex set of a cycle**. The vertex set $Ver(C)$ of a cycle C is the set of all the vertices of C.

- **Edge set of a cycle**. The edge set $Edge(C)$ of a cycle C is the set of all the edges of C.

- **Degree**. The degree $d(v)$ of a vertex v is the number of adjacent edges to v in a line drawing.
- **Simple line drawing**. A line drawing is called simple if there exists no internal face in the manifold that the line drawing represents.
- **Connected cycles**. Two cycle C_a and C_b are called connected if $Ver(C_a) \cap Ver(C_b) \neq \emptyset$.
- **Connected edge and cycle**. An edge $e = \{v_{e1}, v_{e2}\}$ is called connected to a cycle C if $\{v_{e1}, v_{e2}\} \cap Ver(C) \neq \emptyset$ and $e \notin Edge(C)$.
- **Neighboring real faces**. Two real faces f_a and f_b are called neighboring if there exists an edge e such that $e \in Edge(f_a) \cap Edge(f_b)$, and f_a and f_b are called the neighboring real faces of e.
- **Partition of a set**. Given a non-empty set S, a partition $P_S = \{S_1, S_2\}$ is a set of two non-empty subsets S_1 and S_2 of S such that $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$.

Many of these terms are illustrated with the line drawings in Fig. 2.2. Now we explain why to have Assumption 2.4. An internal face is where two separated manifolds are glued together. An internal face may be non-planar. However, we treat all the internal faces as planar in this chapter. This is true for most of practical objects with internal faces. The advantage of this treatment is that when an object is separated along an internal face, this internal face becomes a real planar face.

2.3 Separation of a Line Drawing

There are many ways to partition the edge-vertex graph of a line drawing into multiple smaller graphs. However, these graphs are meaningless if they do not represent real objects. Obviously, it is desirable that each of the separated line drawings still represents a manifold. We use this as a requirement to design a method

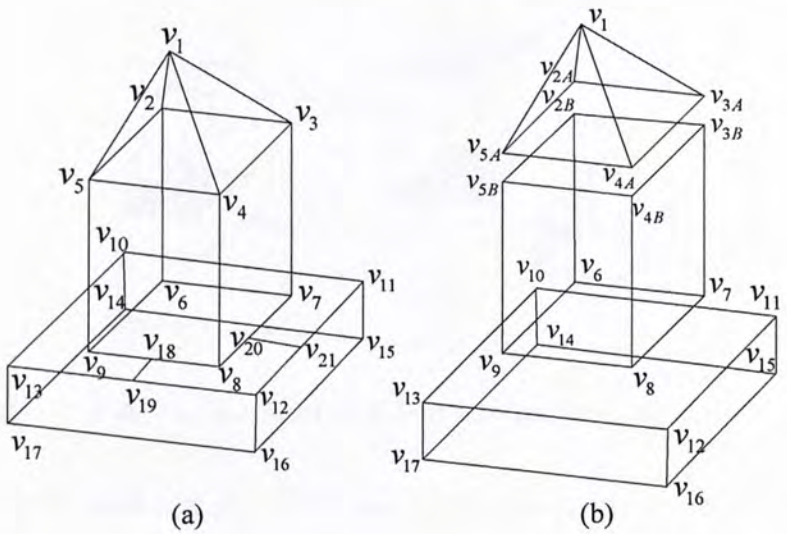


Fig. 2.2: Illustration of some terms. (a) Cycle (v_1, v_3, v_4, v_1) is a real face. Cycles $(v_2, v_3, v_4, v_5, v_2)$ and $(v_6, v_7, v_{20}, v_8, v_{18}, v_9, v_6)$ are two internal faces. Edges $\{v_{18}, v_{19}\}$ and $\{v_{20}, v_{21}\}$ are two artificial lines indicating the coplanarity of cycles $(v_6, v_7, v_8, v_9, v_6)$ and $(v_{10}, v_{11}, v_{12}, v_{13}, v_{10})$. Edge $\{v_1, v_4\}$ is a chord of cycle $(v_1, v_5, v_4, v_3, v_1)$. Two real faces $(v_2, v_3, v_7, v_6, v_2)$ and (v_1, v_3, v_4, v_1) are connected. Edge $\{v_3, v_7\}$ and face (v_1, v_2, v_3, v_1) are connected. (b) The line drawing in (a) is separated into three simple line drawings.

for line drawing separation. By observing numerous complex objects, especially man-made objects, we can see that most of them are formed by gluing two or more smaller objects together, resulting in internal faces. Therefore, our strategy is to find the internal faces from a line drawing first and then separate it along the internal faces.

2.3.1 Classification of Internal Faces

An internal face is where two separated manifolds are glued together. Fig. 2.3 shows four internal faces. An internal face may be non-planar. However, we treat all the internal faces as planar in this chapter. This is true for most of the practical objects with internal faces. The advantage of this treatment is that when an object is separated along an internal face, this internal face becomes a real planar face.

Let f_1 and f_2 be two real faces in two separated manifolds that are glued together

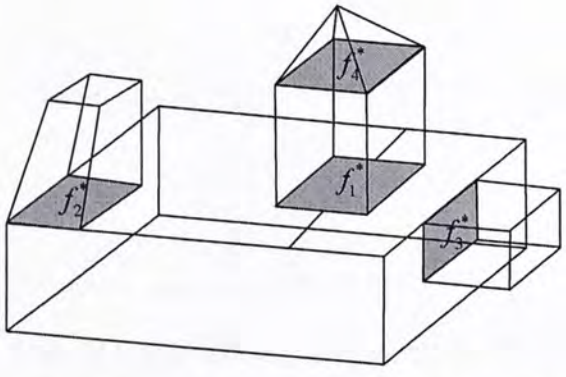


Fig. 2.3: Illustration of four internal faces f_{1-4}^*.

generating an internal face f^*. Let C_1 and C_2 be the two cycles corresponding to f_1 and f_2, respectively, in the original line drawing. We can classify f^* into one of the two types: 1) C_1 and C_2 have no contact, and 2) C_1 and C_2 have contact (partly or completely). In Fig. 2.3, f_1^* belongs to type 1, and f_2^*, f_3^*, and f_4^* belong to type 2. Note that for f_4^*, C_1 and C_2 merge into one in the line drawing.

2.3.2 Separating a Line Drawing along Internal Faces of Type 1

When f^* belongs to type 1, since C_1 and C_2 are not touched, additional information must be used to indicate the coplanarity of C_1 and C_2 so that correct face identification and reconstruction from the line drawing are possible. Using artificial lines to indicate this coplanarity is the simplest and most straightforward way, which has been used in solid modeling [3], [52]. Two artificial lines connecting two edges of C_1 to two edges of C_2 are added by the user who draws the line drawing[2] . For internal faces of type 1, if we can detect the related artificial lines and remove them, then we can separate the line drawing along these internal faces. The following proposition is for this purpose.

[2]Note that one artificial line is not enough to indicate the coplanarity. According to Proposition 2.1, we can find and remove artificial lines, as shown in Fig. 2.2. From Fig. 2.2(b), $C_1 = (v_6, v_7, v_8, v_9, v_6)$ and $C_1' = (v_8, v_9, v_{5B}, v_{4B}, v_8)$ are both identified as real faces from the middle line drawing. With only $\{v_{18}, v_{19}\}$, we cannot know if C_1 or C_1' is the internal face when both C_1 and C_1' are enclosed by the cycle $(v_{10}, v_{11}, v_{21}, v_{12}, v_{19}, v_{13}, v_{10})$ in the original line drawing in another viewpoint. However, we know that it is C_1 with both $\{v_{18}, v_{19}\}$ and $\{v_{20}, v_{21}\}$.

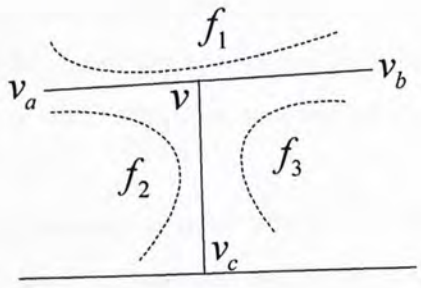


Fig. 2.4: Part of a line drawing with an artificial line $\{v, v_c\}$.

Proposition 2.1 Let $\{v, v_a\}$, $\{v, v_b\}$, and $\{v, v_c\}$ be the three edges connected to a vertex v of degree 3. If $\{v, v_a\}$ and $\{v, v_b\}$ are collinear, then $\{v, v_c\}$ is an artificial line.

Proof. Assume, to the contrary, that $\{v, v_c\}$ is not an artificial line but an edge, as shown in Fig. 2.4. Since the line drawing denotes a manifold, every edge is passed through by two faces and hence three faces f_1, f_2 and f_3 pass through v (see Fig. 2.4). According to the assumption that the line drawing is the projection of a manifold in a generic view, the three vertices v_a, v, and v_b are also collinear in 3D space. Thus, the straight line $\overline{v_a v v_b}$ and vertex v_c define a plane in 3D space, implying that f_2 and f_3 are coplanar, which contradicts the definition that an edge is the intersection of two non-coplanar real faces. Therefore, $\{v, v_c\}$ is an artificial line. ■

With Proposition 2.1, all the artificial lines can be detected and removed. Note that when an artificial line is removed, its two vertices in the original line drawing are also removed. For the example in Fig. 2.4, the two collinear edges $\{v, v_a\}$ and $\{v, v_b\}$ become one edge $\{v_a, v_b\}$ after v is removed. An internal face of type 1 turns out to be a real face in the separated line drawing. The face $(v_6, v_7, v_8, v_9, v_6)$ in Fig. 2.2(b) is such an example.

2.3.3 Detecting Internal Faces of Type 2

Even with the real faces known from a line drawing (Assumption 2.3), detecting internal faces of type 2 is not a trivial problem. In this chapter, the detection is

performed through a cycle-searching scheme. Since exhaustive searching is computationally expensive, we here develop some properties related to internal faces of type 2 so that most cycles that cannot be such an internal face can be eliminated during the search.

In this section, we only consider internal faces of type 2. For concision, we simply use "internal face(s)" to denote "internal face(s) of type 2". When we say two cycles overlap, we mean that their enclosed regions overlap on the 2D line drawing plane. The first two properties below come from the definition and observation of common internal faces. They are useful to develop other subsequent properties. Assumption 2.4 (all internal faces are planar) is also implied.

Property 2.1 An internal face except its boundary is invisible.

Property 2.2 Two coplanar internal faces do not overlap with each other.

Property 2.3 A self-intersecting cycle is not an internal face.

Proof. Since the projection of the boundary of a planar face cannot form a self-intersecting cycle [52], an internal face, which is formed by gluing the real faces of two planar manifolds, cannot be self-intersecting either. ■

Property 2.4 If two cycles share two or more non-collinear edges and overlap with each other, then they cannot both be internal faces.

Proof. Since the two cycles share two or more non-collinear edges, they must be coplanar if they are internal faces. From Property 2.2, they cannot both be internal faces. ■

Property 2.5 A cycle cannot be an internal face if the cycle has a chord that is completely or partially enclosed inside the cycle.

Proof. The chord is obviously on the same plane with the cycle if the cycle is an internal face (see Fig. 2.5). An edge of a line drawing lies on the surface of the

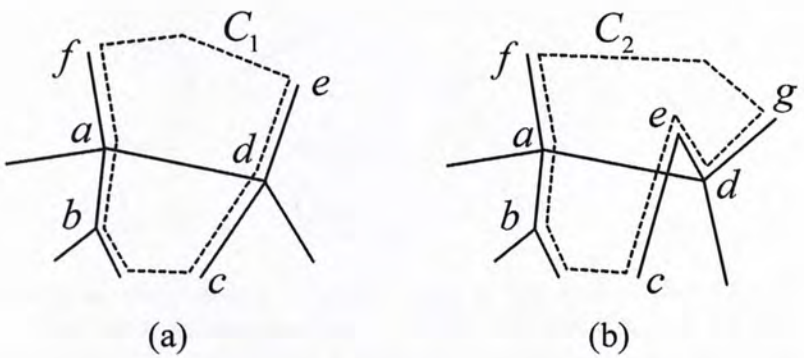


Fig. 2.5: (a) A cycle $C_1 = (f, a, b, ..., c, d, e, ...f)$ with a chord $\{a, d\}$ enclosed completely by it. (b) Another cycle $C_2 = (f, a, b, ..., c, e, d, g, ...f)$ with a chord $\{a, d\}$ enclosed partially by it.

manifold and is visible from a certain viewpoint in 3D space. Hence, it or part of it cannot be an interior part of an internal face. By Property 2.1, we have this property. ■

Property 2.6 A cycle cannot be an internal face if this cycle and a real face share two or more non-collinear edges and they have an overlapping region.

Proof. If this cycle is an internal face and shares two or more non-collinear edges with a real face, then the cycle and the real face lie on the same plane. If they further have an overlapping region in the line drawing, then this region is visible, which contradicts Property 2.1. ■

Property 2.7 A cycle cannot be an internal face if this cycle and a real face share two or more non-collinear edges and they have intersecting edges.

Proof. If this cycle is an internal face and shares two or more non-collinear edges with a real face, then the cycle and the real face lie on the same plane. If they further have intersecting edges, then part of the real face must be enclosed by the internal face, which contradicts Property 2.1. ■

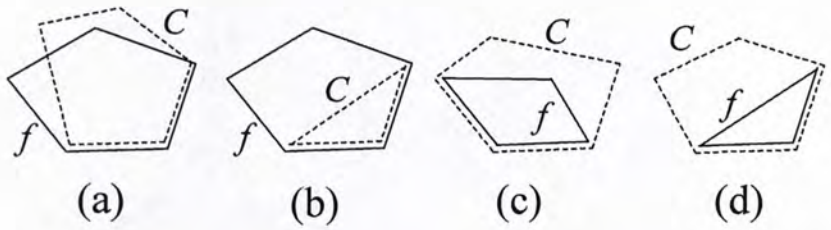


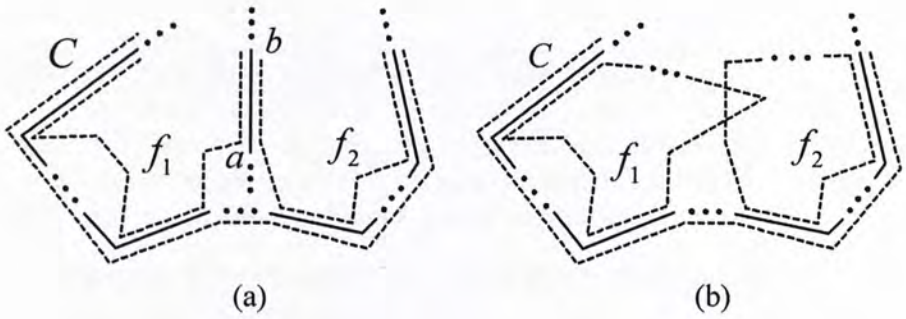
Fig. 2.6: Several cases where a cycle C and a real face f share two non-collinear edges and they have an overlapping region in the line drawing. (a) C and f have intersecting edges. (b)–(d) C and f have an overlapping region without intersecting edges.

Fig. 2.6 shows several cases where a cycle and a real face share two non-collinear edges and they have an overlapping region in the line drawing. In fact, Property 2.7 is a special case of Property 2.6. It is stated explicitly as a separate property because it can be used to reduce fruitless search for internal faces before a path becomes a cycle. For other cases such as Figs. 2.6(b)–(d) where the cycle and the real face have no intersecting edges, the cycle can be determined not to be an internal face after the cycle is formed.

Property 2.8 A cycle cannot be an internal face if 1) this cycle shares two or more non-collinear edges with two real faces and 2) the two real faces share an edge or they have an overlapping region.

Proof. Let the cycle, the two real faces be C, f_1, and f_2, respectively. Assume, to the contrary, that C is an internal face. From the first condition, we know that C, f_1, and f_2 all lie on the same plane. If f_1 and f_2 share an edge, as shown in Fig. 2.7(a), the definition that an edge is the intersection of two non-coplanar real faces is violated. In another case, f_1 and f_2 overlap, as shown in Fig. 2.7(b). However, two co-planar real faces of an manifold must not overlap in the line drawing. Therefore, C cannot be an internal face. ■

With these properties, we can develop an algorithm to detect the internal faces of a line drawing, which is summarized in Algorithm 1). It is a depth-first search



(a) (b)

Fig. 2.7: A cycle C sharing two non-collinear edges with two real faces f_1 and f_2. (a) f_1 and f_2 having a common edge $\{a, b\}$. (b) f_1 and f_2 overlapping.

algorithm with the properties incorporated to guide the search. The properties can cut most fruitless branches during the search, and thus considerably speed up the algorithm.

In the algorithm, an array *Path* is used to keep the vertices in the current search path; the variable *index* gives the position of the last-added vertex in *Path* during the search. A binary label *Label*(v) is given for every vertex $v \in \mathcal{V}$ to denote whether the vertex v is in *Path* or not. In practical applications, we can often set the maximum length D_{max} of internal faces to avoid fruitless search. D_{max} is used with a 2D array *Shortest* in step 8(a), which indicates the shortest path length between any two vertices.

The algorithm starts the search from every edge in the line drawing (see step 2), and calls the procedure *INTERNAL* recursively to detect possible internal faces. Obviously, this search does not miss any internal faces. In addition, all our experiments show that the outputted cycles are indeed internal faces, which suggests that the properties can not only speed up the search but also effectively distinguish internal faces from other cycles.

For some line drawings, there exist incompatible internal faces, resulting in multiple solutions from a line drawing (see step 3 and "Output" in Algorithm 1). One example is shown in Fig. 2.8, which has 15 real faces. From this line drawing, Algorithm 1) finds three internal faces $C_1 = (1, 2, 3, 4, 5, 6, 7, 8, 1)$, $C_2 = (1, 2, 3, 4, 5, 8, 1)$,

Algorithm 1 Depth-first search of internal faces of type 2.

Input: An line drawing $\mathcal{L} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ where $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$ are the sets of vertices, edges, and real faces, respectively, the adjacent lists $AdjList(v)$ for every vertex $v \in \mathcal{V}$, a 2D array $Shortest(v_1, v_2)$, $v_1, v_2 \in \mathcal{V}$, indicating the shortest path length between v_1 and v_2, and the maximum search depth D_{max}.

1. Initialization: $\mathcal{F}^* \leftarrow \emptyset$; $Label(v) \leftarrow 0$, for every vertex $v \in \mathcal{V}$;
2. **for** every edge $\{u, v\} \in \mathcal{E}$ **do**
   (a) $index \leftarrow 2$;
   (b) $Path(0) \leftarrow v$; $Path(1) \leftarrow u$;
   (c) **for** every vertex $w_1 \in AdjList(u)$ and $w_1 \neq v$ **do** $INTERNAL(u, w_1)$;
   (d) $Path(0) \leftarrow u$; $Path(1) \leftarrow v$;
   (e) **for** every vertex $w_2 \in AdjList(v)$ and $w_2 \neq u$ **do** $INTERNAL(v, w_2)$;
3. Detect whether there are incompatible internal faces in $\mathcal{F}^*$ according to Property 2.4;

Output: The set of internal faces in $\mathcal{F}^*$ if there are no incompatible internal faces, or the sets of internal faces if there are incompatible internal faces.

procedure $INTERNAL(u, v)$

4. $index \leftarrow index + 1$; $Label(v) \leftarrow 1$;
5. $Path(index) \leftarrow v$;
6. Check whether edge $\{u, v\}$ intersects any previous edges in $Path$ (according to Property 2.3). If yes, **goto** 9;
7. With the set of real faces that shares two or more edges with the current path in $Path$, check whether this path can form an internal face according to Properties 2.7 and 2.8. If no, **goto** 9;
8. **for** every vertex $w \in AdjList(v)$ and $w \neq u$ **do**
   (a) **if** $Label(w) = 0$ and $index + Shortest(Path(0), w) \leq D_{max}$, **then** extend the path by calling $INTERNAL(v, w)$;
   (b) **else if** $w = Path(0)$ (a cycle is obtained in this case) **then**
      i. With all the chords of the cycle in $Path$, check whether this cycle can form an internal face according to Property 2.5. If no, **goto** 9;
      ii. With all the real faces, check whether this cycle can form an internal face according to Property 2.6. If no, **goto** 9;
      iii. Put this cycle into $\mathcal{F}^*$ if it is not a real face and is not in $\mathcal{F}^*$ yet;
9. $index \leftarrow index - 1$; $Label(v) \leftarrow 0$;

end of $INTERNAL$.

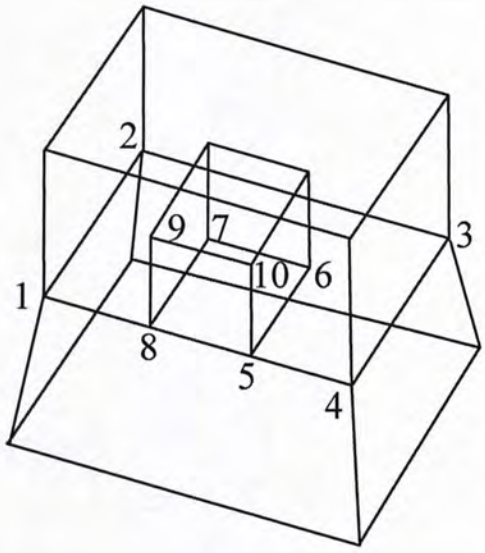


Fig. 2.8: An example where multiple solutions exist.

and $C_3 = (5, 8, 9, 10, 5)$. Since C_1 and C_2 are incompatible according to Property 2.4, the algorithm finally outputs two solutions: one is C_1 and C_3 and the other is C_2 and C_3. Note that when C_1 is an internal face, C_1 and the real face $(5, 6, 7, 8, 5)$ are on the same plane; when C_2 is an internal face, C_2 and this real face are on different planes.

The reader may wonder why C_3 in Fig. 2.8 is also an internal face. In fact, holes and caves may also generate internal faces, but the definition of internal faces needs to be extended a little. Here a hole is one that passes through the surface of a manifold while a cave does not. The manifold in Fig. 2.1 has a hole while the manifold in Fig. 2.8 has a cave. Next we discuss this extension with two simple objects in Figs. 2.9(a) and (c).

In Section 2.2, we define an internal face as a face inside a manifold only with its edges visible on the surface, and it is formed by gluing two manifolds together. One such example is shown in Fig. 2.9(a) where C_1 (denoted by the bold cycle) is the internal face. Obviously, the object in Fig. 2.9(a) can be considered as the union (gluing) of the two smaller objects in Fig. 2.9(b). In another case, the object shown in Fig. 2.9(c) can be considered as the subtraction of the smaller object from

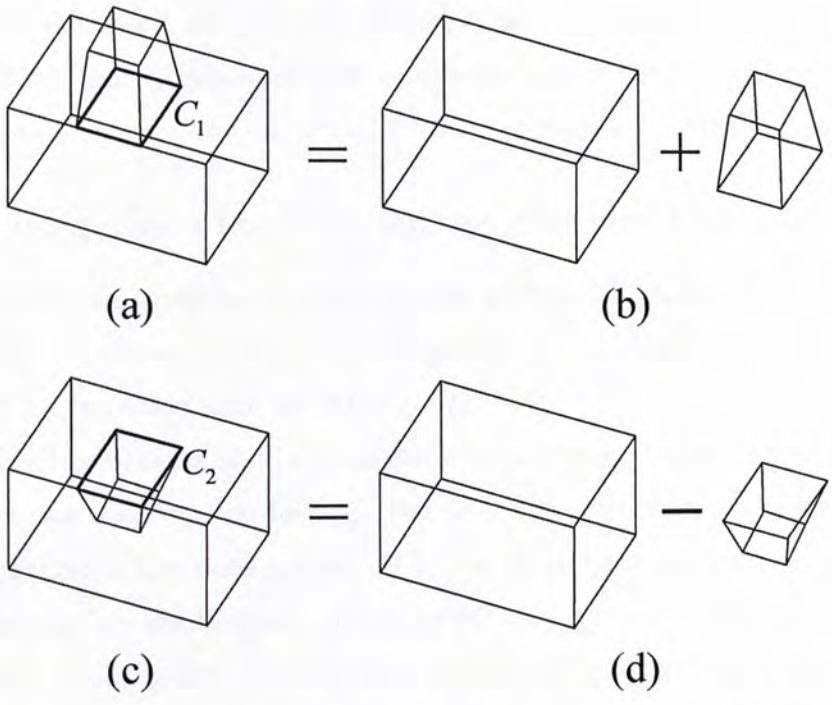


Fig. 2.9: Union and subtraction that generate internal faces. (a) A manifold. (b) Two manifolds whose union can result in the manifold in (a). (c) Another manifold. (d) Two manifolds, with which the subtraction of the smaller one from another can result in the manifold in (c). The two internal faces are C_1 and C_2.

the bigger object shown in Fig. 2.9(d). Comparing the two line drawings (a) and (c), we can see that they have very similar structures and the same topology. The region enclosed by C_1 is invisible and is the common part of the two separate objects in Fig. 2.9(b). There is also such a region enclosed by C_2 (the bold cycle) that is invisible since it is not a real face, and is the common part of the two objects in Fig. 2.9(d). Therefore, we also call C_2 an internal face. Note that Algorithm 1) can also find it as an internal face because Properties 2.1–2.8 do not prevent it from being so. Now we see that an internal face can be formed either by gluing two manifolds together or by cutting a manifold from another. On the other hand, it is easy to know that gluing two manifolds or cutting a manifold from another may not necessarily result in an internal face.

The cycle $C_3 = (5, 8, 9, 10, 5)$ in Fig. 2.8 is similar to the cycle C_2 in Fig. 2.9(c).

In Fig. 2.8, since C_3 is compatible with both C_1 and C_2 while C_1 and C_2 are incompatible, Algorithm 1) obtains the two solutions. Both of them are valid but lead to different separations of the line drawing, which is discussed in the next section.

2.3.4 Separating a Line Drawing along Internal Faces of Type 2

In this section, we again use "internal face(s)" to denote "internal face(s) of type 2" for concision. From an internal face, we separate the line drawing by recovering the two touching faces that form the internal face.

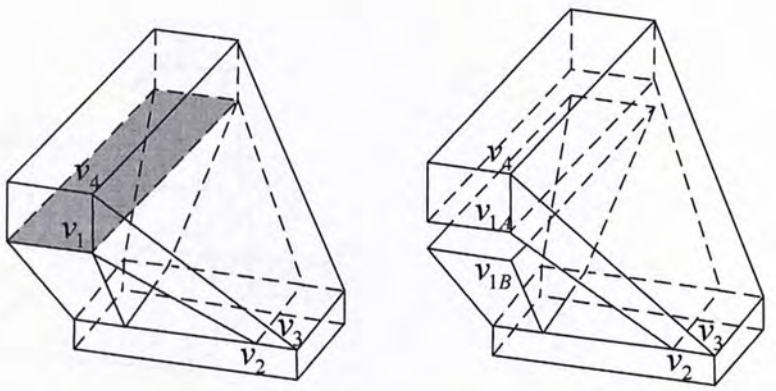
Given a line drawing and its identified real and internal faces, it is not a trivial problem to separate the line drawing. The main difficulties are: 1) the 3D geometry of the manifold is not available yet; 2) in the 2D projection, the lines connecting to an internal face can be in any direction with respect to the internal face; and 3) when a line drawing is separated into two parts along an internal face, for a line that is connected to the internal face in the original line drawing, it is not obvious to which part this line should be connected. For example, the correct separation of the line drawing in Fig. 2.10(a) is given in Fig. 2.10(b). If the edge $\{v_1, v_2\}$ is not connected to v_{1A} but to v_{1B}, a wrong separation then results. It is wrong because the face $(v_1, v_2, v_3, v_4, v_1)$ is broken after such a separation.

Through the observation of different line drawings, we find that the human separation of a line drawing along an internal face f^* always satisfies two conditions:

Condition 2.1 All the real faces connected to f^* are partitioned into two sets, $\mathcal{F}_0(f^*)$ and $\mathcal{F}_1(f^*)$.

Condition 2.2 Two real faces sharing a common edge connected to f^* (not including the edges of f^*) both appear in either $\mathcal{F}_0(f^*)$ or $\mathcal{F}_1(f^*)$.

Condition 2.1 guarantees that each real face connected to f^* is not broken. Condition 2.2 implies that two real faces sharing an edge that is connected to f^* always appear in the same face set $\mathcal{F}_0(f^*)$ or $\mathcal{F}_1(f^*)$. Along all the internal faces of type 2 in the line drawings in Fig. 2.3 and Fig. 2.10(a), our intuitive separations of the



(a) (b)

Fig. 2.10: An example of separating a line drawing along an internal face. (a) The original line drawing with the internal face marked. (b) The correct separation. The hidden edges are shown in dashed for easier observation.

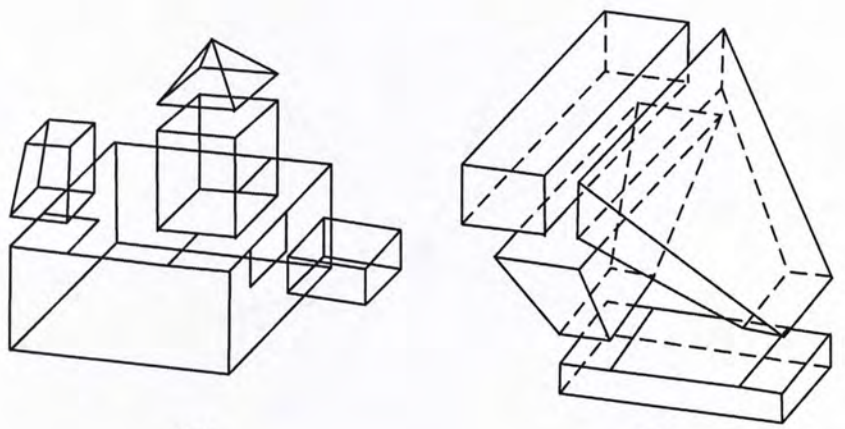
line drawings are shown in Fig. 2.11. We can verify that all these separations satisfy the two conditions above. Mathematically, we formulate such a separation in the following definition, and call it a partition along an internal face.

Definition 2.1 Let f^* be an internal face, $\mathcal{F}(f^*) = \{f_i\}_{i=1}^m$ be the set of all the m real faces connected to f^*, and $\mathcal{E}(f^*) = \{e_i\}_{i=1}^n$ be the set of all the n edges connected to f^*. A **partition along** f^* is to find a face set partition $P_{\mathcal{F}(f^*)} = \{\mathcal{F}_0(f^*), \mathcal{F}_1(f^*)\}$ and an edge set partition $P_{\mathcal{E}(f^*)} = \{\mathcal{E}_0(f^*), \mathcal{E}_1(f^*)\}$ simultaneously such that for any $e \in \mathcal{E}_s(f^*)$, it holds that $e \notin Edge(f), \forall f \in \mathcal{F}_{1-s}(f^*)$, where $s = 0, 1$. This partition along f^* is denoted by $P_{f^*} = (P_{\mathcal{F}(f^*)}, P_{\mathcal{E}(f^*)})$.

We can prove that a partition along an internal face has the following properties.

Property 2.9 Given a partition $P_{f^*} = (P_{\mathcal{F}(f^*)}, P_{\mathcal{E}(f^*)})$ along an internal face f^*, for two real faces $f_a, f_b \in \mathcal{F}(f^*)$, if $f_a \overset{\mathcal{E}(f^*)}{\sim} f_b$, then both f_a and f_b are in $\mathcal{F}_0(f^*)$ or in $\mathcal{F}_1(f^*)$.

Proof. Since $f_a \overset{\mathcal{E}(f^*)}{\sim} f_b$, according to Definition 2.2, we can find a series of real faces $f_a = f_0, f_1, \cdots, f_{k-1}, f_k = f_b$ such that $Edge(f_i) \cap Edge(f_{i+1}) \subseteq \mathcal{E}(f^*)$ and $Edge(f_i) \cap Edge(f_{i+1}) \neq \emptyset$, $i = 0, 1, \cdots, k-1$. Obviously, we have $\{f_i\}_{i=0}^k \subseteq \mathcal{F}(f^*)$



(a) (b)

Fig. 2.11: (a) Partitions along f_2^*, f_3^*, and f_4^* in Fig. 2.3. (b) Partitions along all the internal faces in Fig. 2.10(a).

and all these faces are classified into $\mathcal{F}_0(f^*)$ or $\mathcal{F}_1(f^*)$. Suppose, to the contrary, that f_a and f_b are not both in $\mathcal{F}_0(f^*)$ or in $\mathcal{F}_1(f^*)$. Without loss of generality, assume $f_a \in \mathcal{F}_0(f^*)$ and $f_b \in \mathcal{F}_1(f^*)$. Then there must exist two neighboring faces f_j and f_{j+1}, $0 \leq j \leq k-1$, such that $f_j \in \mathcal{F}_0(f^*)$ and $f_{j+1} \in \mathcal{F}_1(f^*)$. By the definition of f_j and f_{j+1}, there exists an edge $e \in \mathcal{E}(f^*)$ satisfying $e \in Edge(f_i) \cap Edge(f_{i+1})$. Without loss of generality, we suppose e is classified into $\mathcal{E}_0(f^*)$ in the partition. Since $e \in Edge(f_i) \cap Edge(f_{i+1})$ and f_j and f_{j+1} belong to different face sets, which violates the definition of line drawing partition. Therefore, Property 2.9 always holds. ■

Property 2.10 In a line drawing partition, for any real face $f \in \mathcal{F}_s(f^*)$ where $s = 0, 1$, its edge set satisfies that $Edge(f) \cap \mathcal{E}(f^*) \subseteq \mathcal{E}_s(f^*)$.

Proof. The property results immediately from the definition of the line drawing partition. ■

In the rest of this section, we will prove that the partition along an internal face is unique and the line drawing(line drawings) after each partition still represents(represent) a manifold(manifolds). First, the following Proposition 2.2 states the uniqueness of a partition and indicates a separation scheme to find such partition

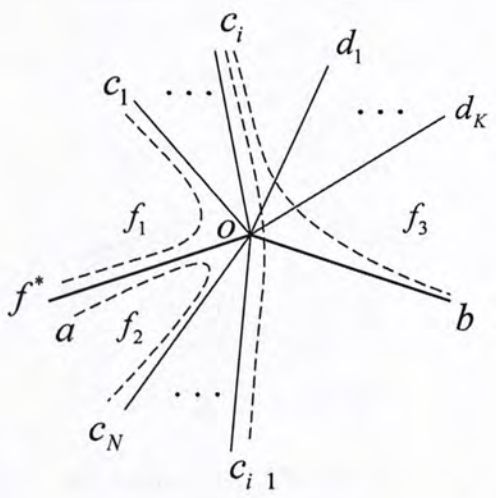


Fig. 2.12: Part of a line drawing where the bold edges and bold dashed edges denote f^* and all the neighborhoods $\mathcal{N}_{v_i}$, $1 \leq i \leq t$, have been stretched into 2D disks

given the line drawing and the internal face which is separated along.

Proposition 2.2 The partition along an internal face of a line drawing representing a manifold exists and is unique.

To prove Proposition 2.2, we shall first introduce three lemmas to describe the topological relationship among all the faces connected to an internal face in the line drawing that represents a manifold.

In order to simplify the mathematical formulations in the lemmas and their proofs, we generalize the term *neighboring real faces* (see Section 2.2) to *$\mathcal{E}$-neighboring real faces*, which is defined as follows.

Definition 2.2 Given a set $\mathcal{E}$ of edges, we call that there exists an **$\mathcal{E}$-path** between real faces f_a and f_b if we can find a series of real faces $f_a = f_0, f_1, \cdots, f_{k-1}, f_k = f_b$ such that $Edge(f_i) \cap Edge(f_{i+1}) \subseteq \mathcal{E}$ and $Edge(f_i) \cap Edge(f_{i+1}) \neq \emptyset$, $i = 0, 1, \cdots, k-1$. These two real faces f_a and f_b are called **$\mathcal{E}$-neighboring**, denoted by $f_a \overset{\mathcal{E}}{\sim} f_b$.

As illustrated in Fig. 2.2, given the edge set $\mathcal{E} = \{\{v_3, v_7\}, \{v_4, v_8\}\}$, the real faces $(v_2, v_3, v_7, v_6, v_2)$ and $(v_4, v_5, v_9, v_8, v_4)$ are $\mathcal{E}$-neighboring. $\mathcal{E}$-neighboring has the following property, the proof of which is trivial and omitted here.

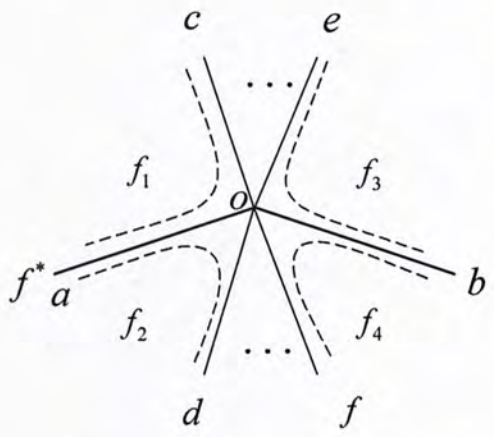


Fig. 2.13: The illustration for Lemma 2.1. The bold solid line represents the internal face passing through ao, ob and the dashed lines denote the real faces.

Property 2.11 $\mathcal{E}$-neighboring is an equivalence relation, which satisfies: 1) reflexivity: $f_a \overset{\mathcal{E}}{\sim} f_a$; 2) symmetry: if $f_a \overset{\mathcal{E}}{\sim} f_b$, then $f_b \overset{\mathcal{E}}{\sim} f_a$; and 3) transitivity: if $f_a \overset{\mathcal{E}}{\sim} f_b$ and $f_b \overset{\mathcal{E}}{\sim} f_c$, then $f_a \overset{\mathcal{E}}{\sim} f_c$.

With the definition above, we now present the three lemmas as follows.

Lemma 2.1 Let e_1 and e_2 be two edges of an internal face f^* joining at vertex o, and let $\mathcal{E}_o$ be the set of all the edges connected to o other than e_1 and e_2. Then, the two real neighboring faces of e_1 are $\mathcal{E}_o$-neighboring to either neighboring face of e_2.

Proof. Suppose the internal face f^* passes through edge ao and ob (corresponding to e_1 and e_2 respectively) as illustrated in Fig. 2.13. The neighboring faces of ao are f_1 and f_2, which pass through oc and od respectively, and the neighboring faces of ob are f_3 and f_4, which pass through oe and of respectively. Now, suppose that f_3 is $\mathcal{E}_o$-neighboring to neither f_1 nor f_2. Then for any pair of points $p_a \in f_3$ and $p_b \in f_1 \cup f_2$, all the paths connecting p_a and p_b must pass through o. We note the neighborhood of point o as $\mathcal{N}_o$. Then according to the conclusion above, $\mathcal{N}_o$ with point o removed is not a connected space. Since the object is a manifold, $\mathcal{N}_o$ should be topologically equivalent to an open disk D. Suppose now we have a homeomorphism $h : \mathcal{N}_o \rightarrow D$. It will also induce a homeomorphism from $h : \mathcal{N}_o - \{o\}$

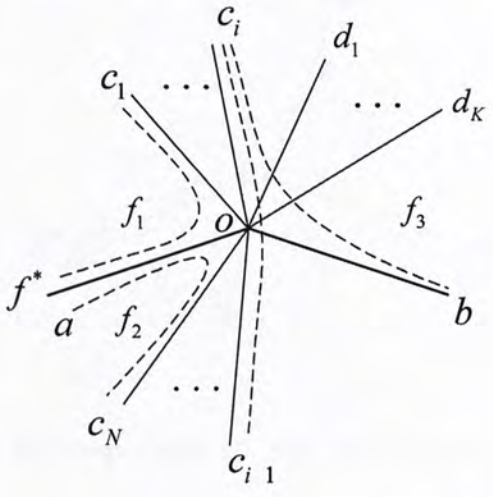


Fig. 2.14: Illustration for Proposition 2.2. $\{oc_1, oc_2, \cdots, oc_N\}$ are the edges which are contained in a series neighboring faces linking f_1 and f_2 while $\{od_1, od_2, \cdots, od_K\}$ refer to unrelated edges.

to $D - h(\{o\})$ but $D - h(\{o\})$ is obviously a connected space [6]. Hence, we know that the neighborhood of o cannot be topologically equivalent to an 2D open disk, which contradicts with the assumption of manifold object. Therefore, f_3 must $\mathcal{E}(f^*)$-neighboring to either f_1 or f_2. The similar result holds for f_4. ■

With the definition of $\mathcal{E}$-neighboring, we can immediately arrive at the following corollary from Lemma 2.1:

Corollary 2.1 Given two arbitrary edges e_1 and e_2 on the internal face f^*, each neighboring face of e_1 is $\mathcal{E}(f^*)$-neighboring to either neighboring face of e_2.

Lemma 2.2 If f_a and f_b are two neighboring faces which are neighboring along the internal face f^*, i.e., their intersection $Edge(f_a) \cap Edge(f_b) \subseteq Edge(f^*)$, then $f_a \overset{\mathcal{E}(f^*)}{\sim} f_b$.

Proof. We assume that the cycle C passes through $\{ao, ob\}$ as Fig. 2.14 illustrates and two neighboring faces f_1 and f_2 of ao passes through the two edges connected with C at vertex o: $\{ao, oc_1\}$ and $\{ao, oc_N\}$ respectively. According to Lemma 2.1

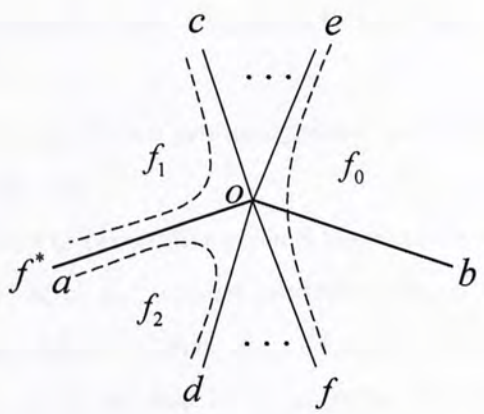


Fig. 2.15: Illustration for Proposition 2.3. The dashed line eof denotes the isolated face.

and Corollary 2.1, the proof of the theorem is equivalent to prove that there exists no series of neighboring faces $\mathcal{F}_C = \{f_{c_1oc_2}, f_{c_2oc_3}, \cdots, f_{c_{N-1}oc_N}\}$ linking f_1 and f_2.

Suppose, to the contrary, that there exists two faces f_a and f_b neighboring along f^* and also $\mathcal{E}(f^*)$-neighboring. And then there exists a series of neighboring faces $\mathcal{F}_C = \{f_{c_1oc_2}, f_{c_2oc_3}, \cdots, f_{c_{N-1}oc_N}\}$ which make f_1 and f_2 $\mathcal{E}(f^*)$-neighboring. We know that the real face f_3 which passes through edge bo are $\mathcal{E}(f^*)$-neighboring to f_1 and f_2 according to Lemma 2.1. Hence, f_3 must pass through the edges in $\mathcal{E}_C = \{oc_1, oc_2, \cdots, oc_N\}$. Without loss of generosity, we let f_3 go through oc_i where $i = 1, 2, \cdots, N$. We will find that the edge oc_i is now shared by 3 real faces: $f_{c_{i-1}oc_i}$, $f_{c_{i-1}oc_{i+1}}$ and f_3, where $f_{c_0oc_1} = f_1$ and $f_{c_Noc_{N+1}} = f_2$. Such result obviously contradicts with the property of the manifold. ■

Lemma 2.3 If f_a and f_b are two arbitrary faces neighboring along the internal face f^*, then for any face f connected with f^*, either $f \overset{\mathcal{E}(f^*)}{\sim} f_a$ or $f \overset{\mathcal{E}(f^*)}{\sim} f_b$ holds.

Proof. Let's consider the case shown in Fig. 2.15 where there exists an isolated face f which connected with f^* at o but $\mathcal{E}(f^*)$-neighboring to neither f_1 nor f_2. Then for any pair of points $p_a \in f$ and $p_b \in f_1 \cup f_2$, all the paths connecting p_a and p_b must pass through o. Similar to the case in Lemma 2.1, the neighborhood of o cannot be topologically equivalent to an 2D open disk and thus it contradicts with

the assumption of manifold object. Therefore, f must $\mathcal{E}(f^*)$-neighboring to either f_1 or f_2. ■

With Lemma 2.1-2.3 above, we can now proceed to the proof of Proposition 2.2.

Proof of Proposition 2.2:

Mathematically, we need to prove that given a line drawing representing a manifold and an internal face f^* of it, Let e be an arbitrary edge of f^*, the real faces f_0 and f_1 be two neighboring faces of e. Then $P_{\mathcal{F}(f^*)}$ defined in Equation 2.1 is a partition of the face set $\mathcal{F}(f^*)$ of all the real faces connected to f^*, and $P_{\mathcal{E}(f^*)}$ defined in Equation 2.2 is a partition of the edge set $\mathcal{E}(f^*)$ of all the edges connected to f^*,

$$\begin{aligned} P_{\mathcal{F}(f^*)} &= \{\mathcal{F}_0(f^*), \mathcal{F}_1(f^*)\} \\ &= \{\{f | f \in \mathcal{F}(f^*), f \overset{\mathcal{E}(f^*)}{\sim} f_s\}\}_{s=0}^{1} \qquad (2.1) \\ P_{\mathcal{E}(f^*)} &= \{\mathcal{E}_0(f^*), \mathcal{E}_1(f^*)\} \\ &= \{\{e | e \in \mathcal{E}(f^*), \exists f \in \mathcal{F}_s(f^*), \\ &\quad s.t. e \in Edge(f)\}\}_{s=0}^{1} \qquad (2.2) \end{aligned}$$

Furthermore, $(P_{\mathcal{F}(f^*)}, P_{\mathcal{E}(f^*)})$ constructs the unique line drawing partition along f^*.

We choose an arbitrary real face $f_0 \in \mathcal{F}(f^*)$ to be the standard face. Let f_1 be the neighboring face of f_0 along the edge of internal face f^*. The edge sets and face sets $\mathcal{E}_0(f^*)$, $\mathcal{E}_1(f^*)$, $\mathcal{F}_0(f^*)$, $\mathcal{F}_1(f^*)$ are defined in Equation 2.1 and Equation 2.2.

First, we will prove that the separation scheme defined in Proposition 2.2 is a line drawing separation. We hence need to prove that the separation scheme satisfies Definition 2.1.

1. $P_{\mathcal{F}(f^*)} = \{\mathcal{F}_s(f^*)\}_{s=0}^{1}$ is a partition of $\mathcal{F}(f^*)$. With Lemma 2.3 and the definition of $\mathcal{F}_0(f^*)$ and $\mathcal{F}_1(f^*)$ in Equation 2.1, it follows immediately that for any real face $f \in \mathcal{F}(f^*)$, either $f \in \mathcal{F}_0(f^*)$ or $f \in \mathcal{F}_1(f^*)$ holds. Hence, we have that $\mathcal{F}(f^*) = \mathcal{F}_0(f^*) \cup \mathcal{F}_1(f^*)$. Then, we need to prove that $\mathcal{F}_0(f^*) \cap \mathcal{F}_1(f^*) = \emptyset$. Suppose, on the contrary, that there exists a real face $f \in \mathcal{F}_0(f^*) \cap \mathcal{F}_1(f^*)$. Then according to the definition of $\mathcal{F}_0(f^*)$, $\mathcal{F}_1(f^*)$, we have $f \overset{\mathcal{E}(f^*)}{\sim} f_0$ and

$f \overset{\mathcal{E}(f^*)}{\sim} f_1$. With the transitivity of $\mathcal{E}(f^*)$-neighboring, it follows that $f_0 \overset{\mathcal{E}(f^*)}{\sim} f_1$, which obviously contradicts Lemma 2.2.

2. $P_{\mathcal{E}(f^*)} = \{\mathcal{E}_s(f^*)\}_{s=0}^1$ is a partition of $\mathcal{E}(f^*)$. For any edge $e \in \mathcal{E}(f^*)$, its two neighboring face f_1 and f_2 are in either $\mathcal{F}_0(f^*)$ or $\mathcal{F}_1(f^*)$. According to the definition of $\mathcal{E}_0(f^*)$ and $\mathcal{E}_1(f^*)$ are defined in Equation 2.2, either $e \in \mathcal{E}_0(f^*)$ or $e \in \mathcal{E}_1(f^*)$ holds, which means that $\mathcal{E}(f^*) = \mathcal{E}_0(f^*) \cup \mathcal{E}_1(f^*)$. Then, we need to prove that $\mathcal{E}_0(f^*) \cap \mathcal{E}_1(f^*) = \emptyset$. Suppose, on the contrary, that there exists a real face $e \in \mathcal{E}_0(f^*) \cap \mathcal{E}_1(f^*)$. Then there exists two real faces $f_a \in \mathcal{F}_0(f^*)$ and $f_b \in \mathcal{F}_1(f^*)$ such that $e \in Edge(f_a) \cap Edge(f_b)$, i.e., $f_a \overset{\mathcal{E}(f^*)}{\sim} f_b$. Then according to the definition of $\mathcal{F}_0(f^*)$, $\mathcal{F}_1(f^*)$, we have $f_a \overset{\mathcal{E}(f^*)}{\sim} f_0$ and $f_b \overset{\mathcal{E}(f^*)}{\sim} f_1$. With the transitivity of $\mathcal{E}(f^*)$-neighboring, it follows that $f_0 \overset{\mathcal{E}(f^*)}{\sim} f_1$, which obviously contradicts Lemma 2.2.

3. $(P_{\mathcal{F}(f^*)}, P_{\mathcal{E}(f^*)})$ satisfy the condition of a line drawing partition. Suppose, on the contrary, there exists an edge $e \in \mathcal{E}_0(f^*)$ and a real face $f \in \mathcal{F}_1(f^*)$ such that $e \in Edge(f)$. With the definition of $\mathcal{E}_0(f^*)$, we know that there exists another face $f_a \in \mathcal{F}_0(f^*)$ such that $e \in Edge(f_a)$. Hence, we have $e \in Edge(f) \cap Edge(f_b)$, i.e., $f \overset{\mathcal{E}(f^*)}{\sim} f_a$. Similar to the proof above, we can finally find that it is contradictory with Lemma 2.2.

As a conclusion, we have proved that the separation scheme defined in Proposition 2.2 in a partition.

Second, we will prove that the line drawing partition is unique when the line drawing and the internal face is given. Suppose that there exist another line drawing partition $(P'_{\mathcal{F}(f^*)}, P'_{\mathcal{E}(f^*)})$ other than the one given in Proposition 2.2, where $P'_{\mathcal{F}(f^*)} = \{\mathcal{F}_s(f^*)'\}_{s=0}^1$ and $P'_{\mathcal{E}(f^*)} = \{\mathcal{E}_s(f^*)'\}_{s=0}^1$. The face set $\mathcal{F}_0(f^*)'$ is still the set containing the standard face f_0. We will then show that $\mathcal{F}_0(f^*) = \mathcal{F}_0(f^*)'$ and $\mathcal{E}_0(f^*) = \mathcal{E}_0(f^*)'$ such that $\mathcal{F}_1(f^*) = \mathcal{F}_1(f^*)'$ and $\mathcal{E}_1(f^*) = \mathcal{E}_1(f^*)'$ can be obtained immediately from the definition of a partition.

1. Showing that $\mathcal{F}_0(f^*) = \mathcal{F}_0(f^*)'$. Since $f_0 \in \mathcal{F}_0(f^*)'$, the fact that $\mathcal{F}_0(f^*) \subseteq$

$\mathcal{F}_0(f^*)'$ results directly from the definition of $\mathcal{F}_0(f^*)$ in Equation 2.1 and Property 2.9. Now we will show that $\mathcal{F}_0(f^*)' \subseteq \mathcal{F}_0(f^*)$. Suppose, on the contrary, that there exists a face $f \in \mathcal{F}_0(f^*)'$ and $f \notin \mathcal{F}_0(f^*)$. Hence, we have $f \in \mathcal{F}_1(f^*)$. According to the definition of $\mathcal{F}_1(f^*)$ in Equation 2.1 and the transitivity of $\mathcal{F}$-neighboring, it follows that $f \overset{\mathcal{E}(f^*)}{\sim} f'$ for any real face $f' \in \mathcal{F}_1(f^*)$, and thus by Property 2.9 we know that $\mathcal{F}_1(f^*) \subseteq \mathcal{F}_0(f^*)'$. On the other hand, we have already shown that $\mathcal{F}_0(f^*) \subseteq \mathcal{F}_0(f^*)'$. As a result, $\mathcal{F}_0(f^*) \cap \mathcal{F}_0(f^*) = \mathcal{F}(f^*) \subseteq \mathcal{F}_0(f^*)'$. Therefore, the only possible case is $\mathcal{F}_0(f^*)' = \mathcal{F}(f^*)$ and it follows that $\mathcal{F}_1(f^*)' = \emptyset$, which is contradictory with the definition of a partition that $\mathcal{F}_1(f^*)' \neq \emptyset$. As a conclusion, we have proved that $\mathcal{F}_0(f^*) = \mathcal{F}_0(f^*)'$.

2. Showing that $\mathcal{E}_0(f^*) = \mathcal{E}_0(f^*)'$. For any edge $e \in \mathcal{E}(f^*)$, we can obtain one of its neighboring faces f. It is obvious that $f \in \mathcal{F}(f^*)$. Hence the classification of f is uniquely determined in any line drawing partition according to the discussion above, in the line drawing partition. By Property 2.10, we know that the classification of e is also uniquely determined. It means that $\mathcal{E}_0(f^*) = \mathcal{E}_0(f^*)'$.

As a conclusion, we have proved that the line drawing partition is unique given the line drawing and the internal face.■

An algorithm can be designed to find such line drawing partition given the internal face f^*. In the beginning, $\mathcal{E}_0(f^*)$, $\mathcal{E}_1(f^*)$ and $\mathcal{F}_1(f^*)$ will be set empty, and $\mathcal{F}_1(f^*)$ will be initialized to contain an arbitrary face $f_0 \in \mathcal{F}(f^*)$. We conclude following three rules from the lemmas proposed above to classify the remaining edges(faces) through an inference from the classified edges and faces and hence augment the edge sets $\mathcal{E}_0(f^*)$, $\mathcal{E}_1(f^*)$ and face sets $\mathcal{F}_0(f^*)$, $\mathcal{F}_1(f^*)$.

Rule 2.1 Given the edge $e \in \mathcal{E}_s(f^*)$ where $s = 0, 1$ and its two neighboring faces are f_1 and f_2, we can infer that $f_1 \in \mathcal{F}_s(f^*)$ and $f_2 \in \mathcal{F}_s(f^*)$.

Rule 2.2 Given the face $f \in \mathcal{F}_s(f^*)$ where $s = 0, 1$ and an edge $e \in Edge(f)$, we can infer that $e \in \mathcal{E}_s(f^*)$.

Rule 2.3 If two faces f_i and f_j are neighboring along the internal face f^* and f_i is classified as $f_i \in \mathcal{F}_s$ where $s = 0, 1$, then we can infer that $f_j \in \mathcal{F}_{1-s}(f^*)$, where $s = 0, 1$.

Proposition 2.2 guarantees that all the edges and faces connected to the internal face f^* can finally be classified by iteratively and alternately applying Rule 2.1, Rule 2.2 and Rule 2.3 as long as the object the line drawing represents is a manifold.

The complete algorithm for separating the object along internal faces is list in Algorithm 2. The separation is on the basis of Proposition 2.2 and three rules proposed above. The duplicate of the vertices $\{v_i'\}_{i=1}^K$ along the internal face f^* and two artificial faces f_0^* and f_1^* which are in the place in which the internal face f^* lies are added to the face set $\mathcal{F}$, forming a new set of line drawing after each separation is performed.

Since there may be more than one internal face in the original line drawing, we hope to further repeat such separation process along other internal faces in the remaining line drawing. For the line drawing in Fig. 2.10(a), four partitions along the four internal faces separate it into four simpler line drawings. This requires that the line drawings generated from the separation process above still represent manifold objects so that the prerequisite of Proposition 2.2 can be met. In view of this, we need the following Proposition 2.3 to guarantee that the remaining line drawings conserve the property of manifold after the separation scheme given in Algorithm 2.

Proposition 2.3 If the initial line drawing $\mathcal{L}$ which represents a manifold object, then after the partition along an internal face, the line drawing (line drawings) still represents (represent) a manifold (manifolds).

Proof. If f^* is a Type 1 internal face, the result is obvious. Here we consider the case that f^* is a Type 2 internal face. Since the separation only change the topological

Algorithm 2 Separating a line drawing along an internal face of type 2.

Input: The 2D line drawing $\mathcal{L} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ and one of its internal face $f^* = \{v_i\}_{i=1}^K$.

1. Find the set $\mathcal{F}(f^*)$ of the faces connected with f^* and the set $\mathcal{E}(f^*)$ of the edges connected with f^*
2. Pick an arbitrary edge e^* from $\mathcal{E}_{f^*}$ and find the two faces f_a and f_b which pass through e^*.
3. Initialization: $\mathcal{F}_0(f^*) = \{f_a\}$, $\mathcal{F}_1 = \{f_b\}$, $\mathcal{E}_0 = \emptyset$, $\mathcal{E}_1(f^*) = \emptyset$
4. Repeat until all the elements in $\mathcal{E}$ and $\mathcal{F}$ have been classified.
   (a) For each edge $e \in \mathcal{E}$, update the face classification $\mathcal{F}_0(f^*)$, $\mathcal{F}_1(f^*)$ according to Rule 2.1.
   (b) For each face $f \in \mathcal{F}$, update the edge classification $\mathcal{E}_0(f^*)$, $\mathcal{E}_1(f^*)$ according to Rule 2.2.
   (c) For each face $f \in \mathcal{F}$, update the face classification $\mathcal{F}_0(f^*)$, $\mathcal{F}_1$ according to Rule 2.3.
5. Split f^* into two faces $f_0^* = f^* = (v_i)_{i=0}^K$ where $v_0 = v_K$ and $f_1^* = (v_i')_{i=0}^K$, where v_i' are the duplicates of v_i, $i = 0, 1, 2, \cdots, K$.
6. Update the line drawing $\mathcal{L}$ to $\mathcal{L}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$.
   (a) Update the vertex set: $\mathcal{V}' = \mathcal{V} \cup \{v_i'\}_{i=0}^K$.
   (b) Update the edge set: $\mathcal{E}' = (\mathcal{E} - \mathcal{E}_1(f^*)) \cup \mathcal{E}_1'(f^*)$, where $\mathcal{E}_1'$ is the duplicated edge set of $\mathcal{E}_1(f^*)$ in which all those edges pass through $v_i (i = 0, 1, \cdots, K)$ now pass through $v_i' (i = 0, 1, \cdots, K)$ instead.
   (c) Update the face set: $\mathcal{F}' = (\mathcal{F} - \mathcal{F}_1(f^*)) \cup \mathcal{F}_1'(f^*) \cup \{f_0^*, f_1^*\}$, where $\mathcal{F}_1'$ is the duplicated face set of $\mathcal{F}_1(f^*)$ in which all those faces pass through $v_i (i = 0, 1, \cdots, K)$ now pass through $v_i' (i = 0, 1, \cdots, K)$ instead.

Output: The line drawing $\mathcal{L}' = (\mathcal{V}', \mathcal{E}', \mathcal{F}')$ after the separation.

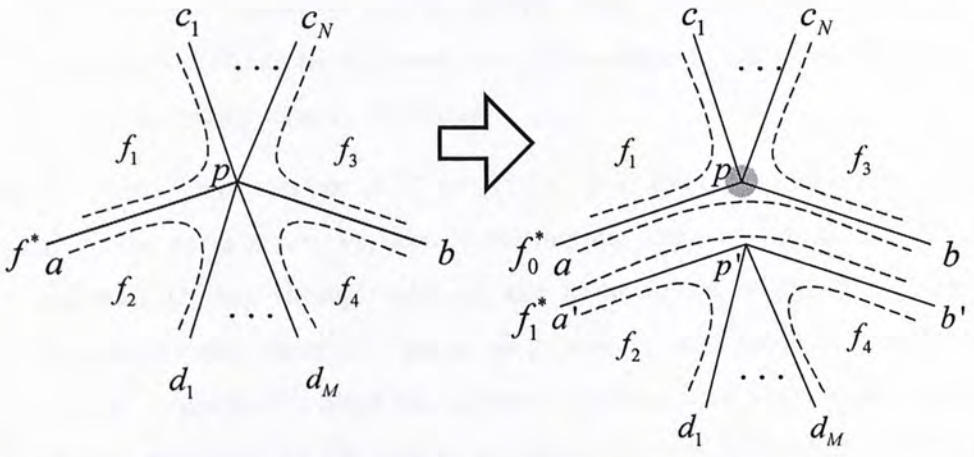


Fig. 2.16: Illustration for Proposition 2.3. The original line drawing is given in the left while the line drawing after the separation is given in the right.

structure of $\mathcal{L}$ along the internal face f^*, in order to prove the manifold property of $\mathcal{L}'$, we thus only need to verify that every point p on the new generated artificial face f_0^* and f_1^* has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space. Now, let us consider the following three cases corresponding to p in the different location of the artificial faces.

1. p is in the interior of the face f_0^* or f_1^*. The proof is trivial in this case due to the planarity assumption of f_0^* and f_1^*.

2. p is on the boundary other than vertices of f_0^* or f_1^*. In this case, we can briefly check whether each edge $e \in Edge(f_0^*) \cup Edge(f_1^*)$ is still shared by exactly two faces so that the neighborhood every point on the boundary edges of new artificial faces have a neighborhood which can be simply transformed into an 2D open disk through folding. In the original line drawing $\mathcal{L}$, for any edge $e_0 \in Edge(f^*)$, we can find two faces $f_a \in \mathcal{F}_0(f^*)$ and $f_b \in \mathcal{F}_1(f^*)$ which share e_0 given the manifold property of $\mathcal{L}$, where $\mathcal{F}_0(f^*)$ and $\mathcal{F}_1(f^*)$ are face sets defined in Algorithm 2. The separation process duplicates the original edge e_0 into two, and we note them as $e_a \in Edge(f_0^*)$ and $e_b \in Edge(f_1^*)$ respectively. From Algorithm 2, we can easily know that e_1 is now shared by face f_a and

f_0^*, and e_2 is shared by face f_b and f_1^*. Since e_0 an arbitrarily edge in the boundary of f^*, we have proved that all the edges in the boundary of f_0^* and f_1^* are shared by exactly two faces.

3. p is one of the vertices of f_0^* or f_1^*, i.e., $p \in Ver(f_0^*) \cup Ver(f_1^*)$. Suppose p is now an arbitrary vertices of the original line drawing $\mathcal{L}$ and the internal face f^* pass through edge ap and pb as shown in Fig. 2.16. We note the neighboring faces of edge ap as f_1 and f_2, and those of edge ob as f_3 and f_4 respectively. Since the original line drawing $\mathcal{L}$ represents a manifold object, according to the results of Lemma 2.1 and Lemma 2.2, we can assume that f_1 is $\mathcal{E}_p$-neighboring to f_3 and f_2 is $\mathcal{E}_p$-neighboring to f_4 without loss of generosity. Hence, there must exist a sequence of neighboring faces $\mathcal{F}_C = \{f_{c_1pc_2}, f_{c_2pc_3}, \cdots, f_{c_{N-1}pc_N}\}$ linking f_1 and f_3, and also another sequence of neighboring faces $\mathcal{F}_D = \{f_{d_1pd_2}, f_{d_2pd_3}, \cdots, f_{d_{M-1}pd_M}\}$ linking f_2 and f_4, where M, N are non-negative integers. After the separation, the internal face f^* is split into two artificial faces $f_0^* = f_{apb}$ and $f_1^* = f_{a'p'b'}$. Now we will verify the neighborhood of both duplicated vertices p and p'. Let us first consider the neighborhood of vertex p. For the verbal simplicity, we define the vertices $c_0 = c_{N+2} = a$ and $c_{N+1} = b$, and also note the set of all faces passing through p as

$$\begin{aligned} \mathcal{F}_p &= \mathcal{F}_C \cup \{f_1, f_3, f_0^*\} \\ &= \{f_{c_0pc_1}, f_{c_1pc_2}, \cdots, f_{c_{N-1}pc_N}, f_{c_Npc_0}\} \end{aligned}$$

Now, we consider the neighborhood of p with a small radius ε can be written as $\mathcal{N}_p^\varepsilon = \{p' | \|pp'\| < \varepsilon \wedge p' \in f, \forall f \in \mathcal{F}_p\}$. Hence, we can define the following piecewise homeomorphism:

$$\begin{aligned} h: \quad & \mathcal{N}_p^\varepsilon \cap f_{c_ipc_{i+1}} \rightarrow \{(r,\theta) : 0 < r < \epsilon, \\ & \frac{2i\pi}{N+2} \leq \theta \leq \frac{2(i+1)\pi}{N+2}\}, i = 0, 1, \cdots, N+1 \end{aligned}$$

h maps the part of $\mathcal{N}_p^\varepsilon$ in each face to a sector in the polar coordinate, and as a result $\mathcal{N}_p^\varepsilon$ is finally mapped into a 2D open disk in Euclidean space. Since

the topological structure at p' is similar to that of p, we can verify that p' has a neighborhood homeomorphic to an 2D open disk in the same way.

From the discussion above, we have shown that the neighborhood of p can be topological equivalent to a 2D open disk in all the cases and thus the line drawing $\mathcal{L}'$ after the separation still represent a manifold object. ■

With Proposition 2.3, such separation process described in Algorithm 2 can be performed iteratively to those separated line drawings until all the internal faces have been split and only simple line drawings are remained.

Now let us see what partition results look like when Algorithm 1) finds multiple solutions from a line drawing. Take the line drawing shown in Fig. 2.8 as an example, in which $\{C_1, C_3\}$ and $\{C_2, C_3\}$ are two solutions with $C_1 = (1, 2, 3, 4, 5, 6, 7, 8, 1)$, $C_2 = (1, 2, 3, 4, 5, 8, 1)$, and $C_3 = (5, 8, 9, 10, 5)$. For convenient observation, it is re-drawn in Fig. 2.17(a).

Consider the first solution $\{C_1, C_3\}$ and suppose that Algorithm 2 does the partition beginning with C_1. Then the result is shown in Fig. 2.17(b). Note that since C_3 has been broken in Fig. 2.17(b), a partition based on it is impossible. Furthermore, since the two line drawings in Fig. 2.17(b) have no internal faces, the result in Fig. 2.17(b) is final. Now suppose that Algorithm 2 starts with C_3 instead of C_1. The first partition result is shown in Fig. 2.17(c). Again C_1 has been broken after the first partition and a partition based on it is impossible. However, From the bigger line drawing in Fig. 2.17(c), Algorithm 1) can find an internal face $C_4 = (1, 2, 3, 4, 1)$. Therefore, Algorithm 2 performs the second partition along C_4 and the final result is given in Fig. 2.17(d).

Consider the second solution $\{C_2, C_3\}$ and suppose that Algorithm 2 carries out the partition along C_2 first. The result is shown in Fig. 2.17(e). Since C_3 is not broken in the upper line drawing in Fig. 2.17(e), further partition along it is performed and the final result is generated as shown in Fig. 2.17(f). If Algorithm 2 starts the partition along C_3 first and then along C_2, the same final result as the one in Fig. 2.17(f) will be obtained.

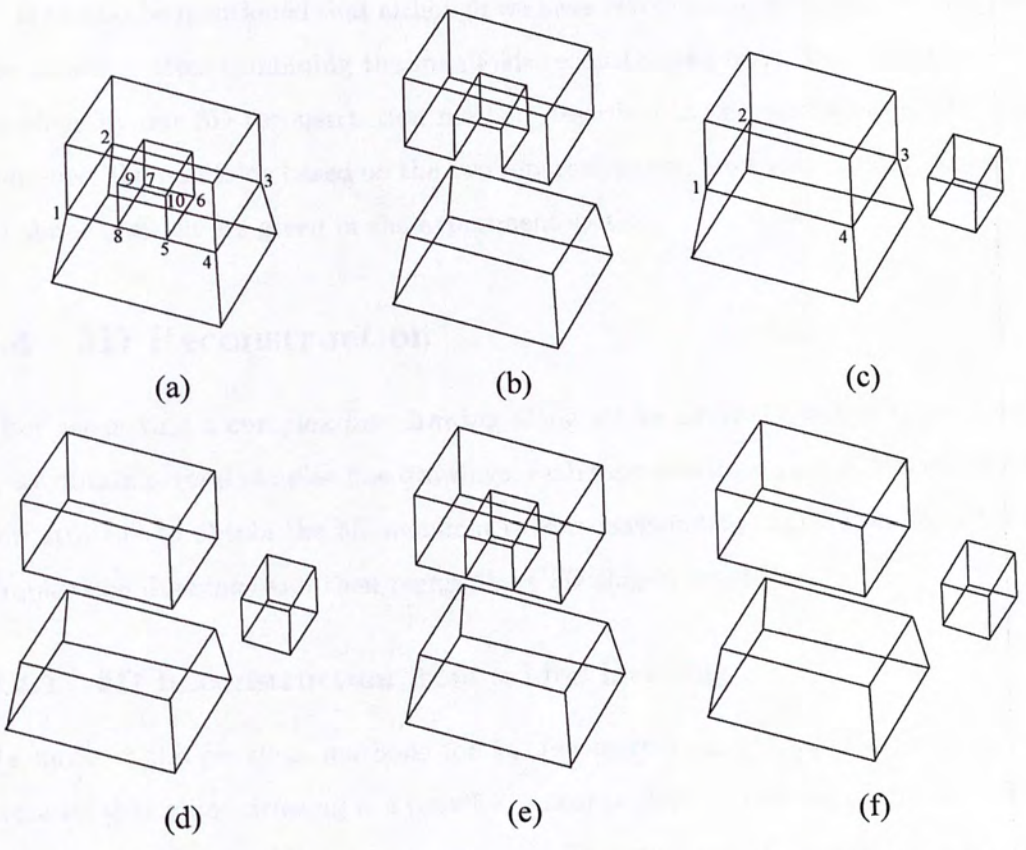


Fig. 2.17: (a) The original line drawing. (b) Partition result along C_1 from (a). (c) Partition result along C_3 from (a). (d) Further partition result along C_4 from (c). (e) Partition result along C_2 from (a). (f) Further partition result along C_3 from (e).

From this example, we can see that even though we have proved that the partition along an internal face is unique (Proposition 2.2), we may obtain multiple separations of a line drawing when the line drawing has more than one internal face. In this example, since the results in Figs. 2.17(d) and (f) are the same, there are only two different separation results as shown in Figs. 2.17(b) and (d).

It should be mentioned that although we have two different separations from this line drawing, after combining the manifolds reconstructed from these separate line drawings by our 3D reconstruction method described in the next section, the two combined 3D manifolds based on the two separations can have very similar expected 3D shapes, which are given in the experiment section.

2.4 3D Reconstruction

After separating a complex line drawing along all its internal faces of types 1 and 2, we obtain several simpler line drawings, each representing a part of the manifold. Our strategy to obtain the 3D manifold is to reconstruct the 3D shapes from these simpler line drawings and then merge these 3D shapes together.

2.4.1 3D Reconstruction from a Line Drawing

As most of the previous methods for 3D reconstruction from a line drawing, we consider that a line drawing is a parallel or near parallel projection of the edges and vertices of a 3D manifold in a generic view. Thus the x- and y-coordinates of each vertex are already known, and only the depth (z-coordinate) needs to be derived. Since the cycles of the real faces are already available too, the surface of the 3D manifold is recovered if the depths of all the vertices are obtained. Among previous methods, the one in [50] can handle 3D reconstruction of most complex objects. For our current problem, however, each separated line drawing is quite simple, and so we can develop a simpler algorithm to handle it.

Reconstructing the 3D shape from each separated line drawing is carried out by an optimization-based approach, in which the objective function contains several

constraints. These constraints try to emulate the human perception of a 2D line drawing as a 3D object. In this chapter, we adopt five constraints: minimizing the standard deviation of angles in the reconstructed object [58], face planarity [43], line parallelism, isometry, and corner orthogonality [49], which are denoted by α_1, α_2, α_3, α_4, and α_5, respectively. The objective function to be optimized is then:

$$f(z_1, z_2, \cdots, z_N) = \sum_{i=1}^{5} \lambda_i \alpha_i, \tag{2.3}$$

where λ_{1-5} are weighting factors, and z_{1-N} are the depths of all the N vertices of a line drawing. We use the hill-climbing method presented in [43] to carry out the minimization.

2.4.2 Merging 3D Manifolds

When all the 3D simple manifolds are available, the next step is to combine them in an appropriate way so that a complete 3D expected object is obtained. The basic idea of our merging process is to well match two manifolds' real faces that correspond to one internal face of the original line drawing.

Suppose that two 3D manifolds O_a and O_b share an internal face f^* with K vertices in the original line drawing, the depths of all O_a's vertices are $z_{a1}, z_{a2}, \cdots, z_{aN_a}$, and the depths of all O_b's vertices are $z_{b1}, z_{b2}, \cdots, z_{bN_b}$. Without lose of generality, also suppose that $z_{a1}, z_{a2}, \cdots, z_{aK}$ are the depths of f^*'s vertices in O_a, and $z_{b1}, z_{b2}, \cdots, z_{bK}$ are the depths of f^*'s vertices in O_b, where z_{ai} corresponds to z_{bi}, $1 \leq i \leq K$. Since O_a and O_b are reconstructed independently, we usually have a large difference between z_{ai} and z_{bi}, $1 \leq i \leq K$, and different sizes of f^* in O_a and O_b. We align them according to the depth means (μ_a and μ_b) and the standard deviations (σ_a and σ_b) of f^* in O_a and O_b, where

$$\mu_j = \frac{1}{K} \sum_{i=1}^{K} z_{ji}, \quad j = a, b, \tag{2.4}$$

$$\sigma_j = \sqrt{\frac{1}{K} \sum_{i=1}^{K} (z_{ji} - \mu_j)^2}, \quad j = a, b. \tag{2.5}$$

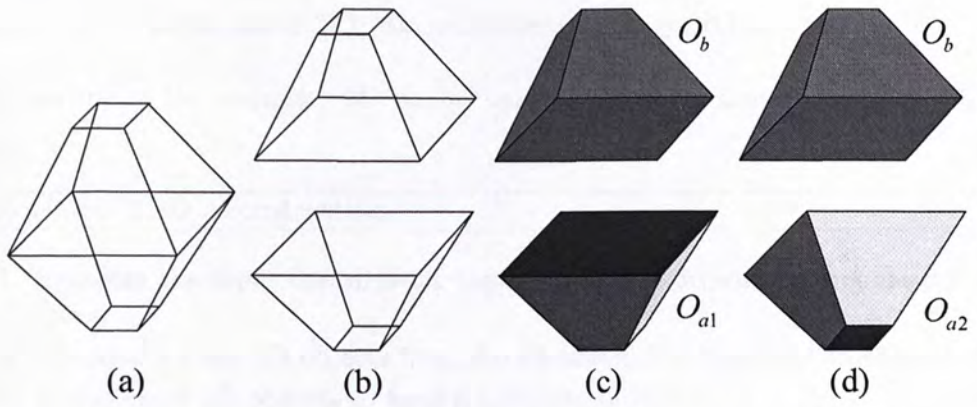


Fig. 2.18: (a) A line drawing. (b) Two separated line drawings. (c) Incompatible objects O_{a1} and O_b. (d) Compatible object O_{a2} and O_b.

While fixing O_b, we modify the depths of all the vertices of O_a by

$$z'_{ai} = \mu_b + \frac{\sigma_b}{\sigma_a}(z_{ai} - \mu_a), \quad i = 1, 2, \cdots, N_a. \tag{2.6}$$

Finally, O_a and O_b are merged by forcing their corresponding vertex depths of f^* to be the same:

$$z''_{ai} = z''_{bi} = \frac{z'_{ai} + z_{bi}}{2}, \quad i = 1, 2, \cdots, K. \tag{2.7}$$

Our visual system can interpret a line drawing as a 3D object in two ways, which is well-known as the Necker cube reversal perception, and this phenomenon also exists in 3D reconstruction from a line drawing [58]. One example is shown in Fig. 2.18 where the lower line drawing in Fig. 2.18(b) may lead to one of the two 3D objects O_{a1} in Fig. 2.18(c) and O_{a2} in Fig. 2.18(d). Incompatible gluing of O_{a1} and O_b happens. To solve this problem, we can turn O_{a1} to O_{a2} by multiplying -1 to all the depths of the vertices of O_{a1}. Before doing this, we need to check if two objects O_a and O_b are compatible. Let

$$s = \text{sgn}\Big(\sum_{i=1}^{K}(z_{ai} - \mu_a)(z_{bi} - \mu_b)\Big). \tag{2.8}$$

If $s = 1$, O_a and O_b are compatible; if $s = -1$, O_a and O_b are not.

2.4.3 The Complete 3D Reconstruction Algorithm

The outline of the complete 3D reconstruction algorithm is summarized in Algorithm 3.

Algorithm 3 3D reconstruction.

1. Separate the input line drawing into simpler line drawings along the internal faces;
2. Reconstruct the 3D objects from the separated line drawings independently;
3. Merge these 3D objects to form a complete object;
4. Fine-tune the complete object.

Steps 1, 2, and 3 have been described in the previous sections. Step 4 is to fine-tune the complete object by performing the minimization of the objective function in (2.3) using the input line drawing. This fine-tuning step can correct some deformation caused by (2.7). The initial values of the depths in this step are the depths of the complete object obtained in step 3. Our experiments show that using step 4 usually generates a better result. When there are multiple separations of a line drawing, as discussed in Section 2.3.4, we can either do the reconstruction from all these separations and output multiple complete manifolds, or just pick any one separation to carry out the 3D reconstruction.

2.5 Experimental Results

In this section, we show a number of examples to demonstrate the performance of our approach. The algorithm is implemented using Visual C++, running on a PC with 3.4 GHz Pentium IV CPU. The weighting factors λ_{1-5} in (2.3) are chosen to be 100, 1, 20, 15, and 20 respectively. These values are obtained by a few experiments first and then fixed in the reconstruction of all the objects.

For some line drawings, there are multiple separations. Let us take the one in Fig. 2.17(a) as an example. Fig. 2.19 shows the two different separations of the line drawing and the reconstruction results based on them. Although the two

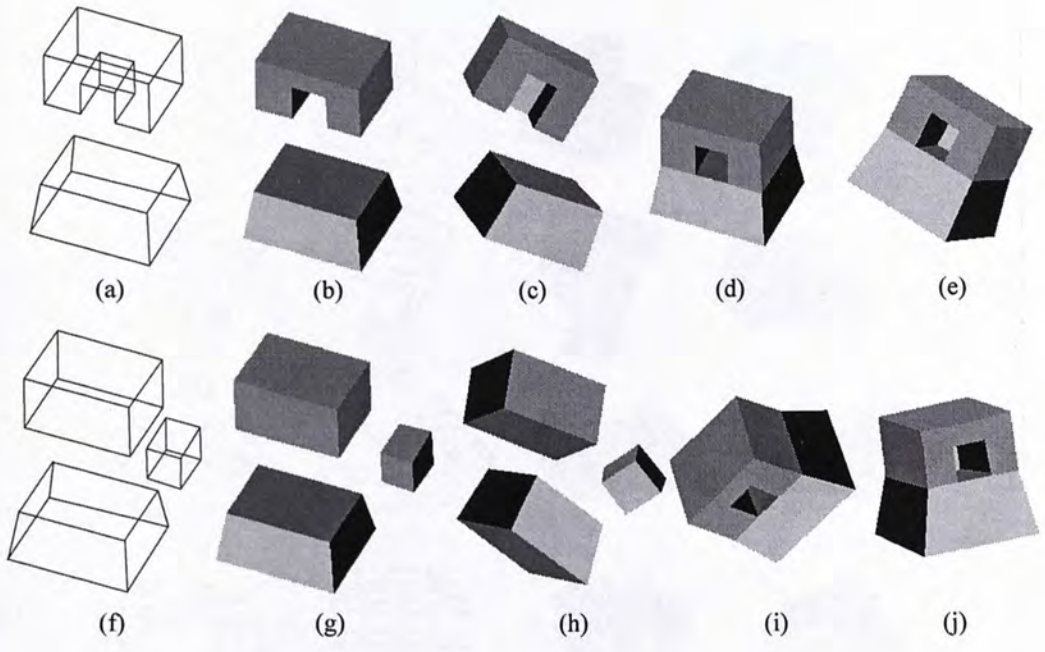


Fig. 2.19: (a) One separation of the line drawing in Fig. 2.17(a). (b) Reconstruction results from the two line drawings in (a). (c) Another view of the results in (b). (d) Result after combining the two manifolds in (b). (e) Another view of the result in (d). (f) Another separation of the line drawing in Fig. 2.17(a). (g) Reconstruction results from the three line drawings in (f). (h) Another view of the results in (g). (i) Result after combining the three manifolds in (g). (j) Another view of the result in (i).

separations are different, the two final combined manifolds are similar and expected. In the following examples, we only show one final result in two views for each line drawing.

Fig. 2.20 shows a set of results and Fig. 2.21 shows another set of more complex results. For each input line drawing, we give a separation result and the reconstructed 3D object displayed in two views, with different grey levels indicating the faces. From Fig. 2.20 and Fig. 2.21, we can see that our algorithm successfully partitions the line drawings and generates expected 3D objects. It is worth mentioning that the objects in Fig. 2.21 are more complex than the objects given in the previous papers for 3D reconstruction from single line drawings.

The method in [50] can do 3D reconstruction of more complex objects than other

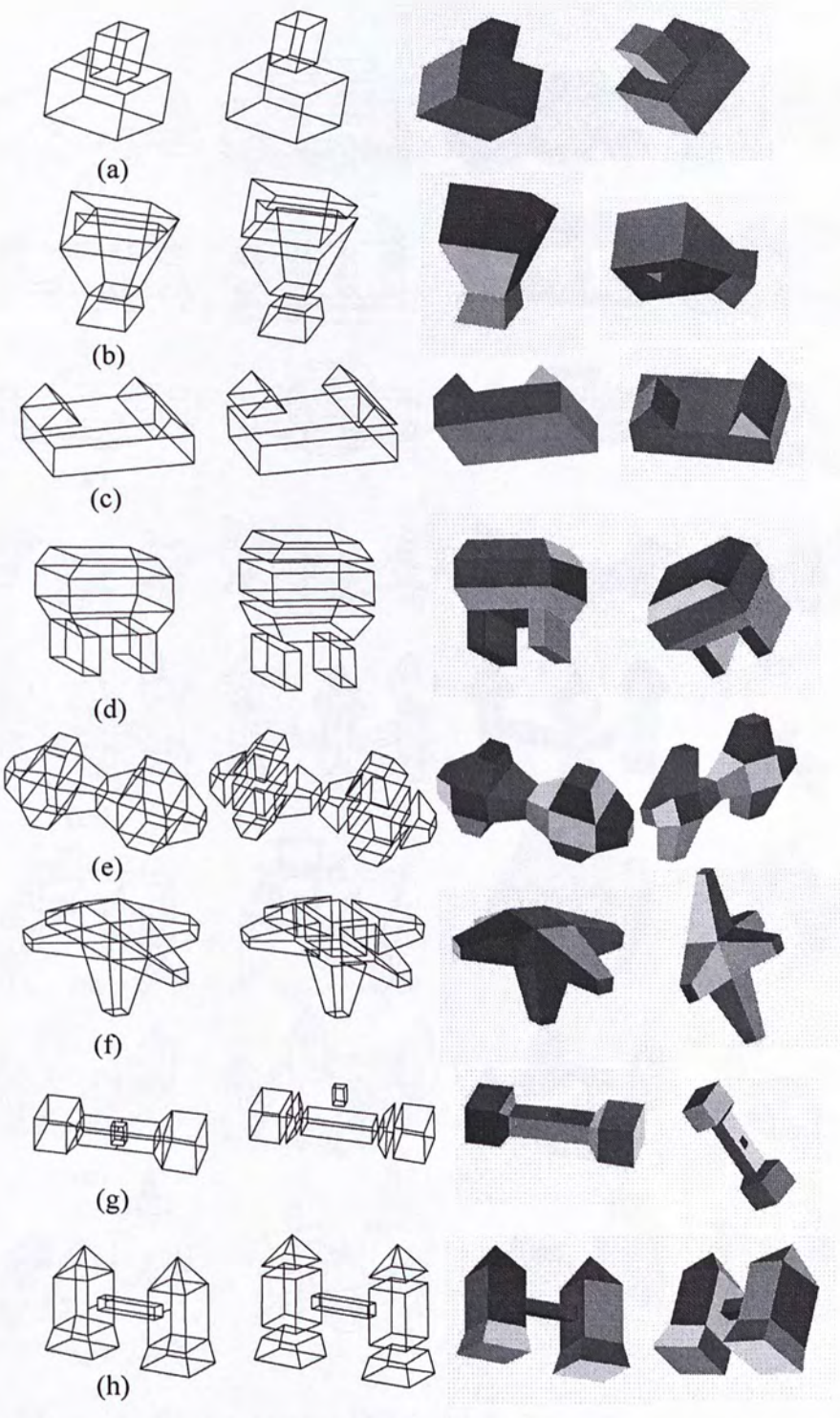


Fig. 2.20: A set of line drawings and their separation and reconstruction results. Different colors are used to indicate the faces.

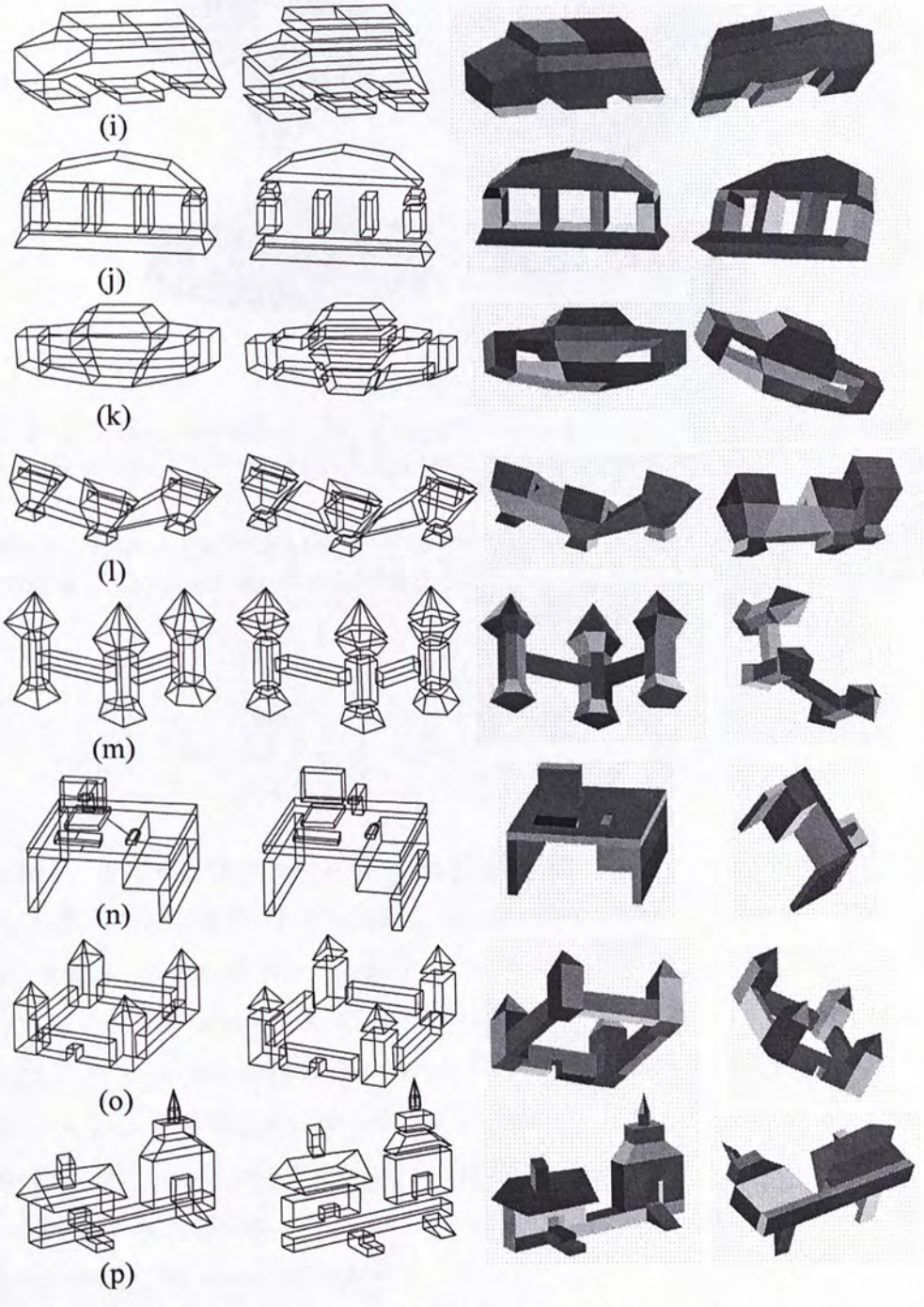


Fig. 2.21: A set of more complex line drawings and their separation and reconstruction results. Different colors are used to indicate the faces.

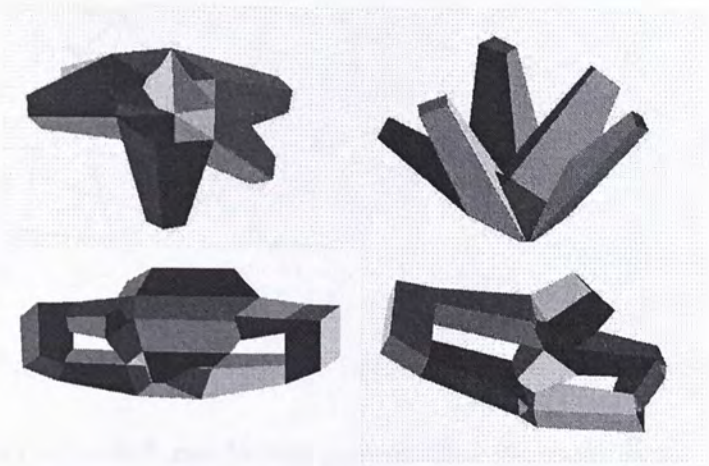


Fig. 2.22: Failed results in two views reconstructed from the 6th line drawing in Fig. 2.20 and 3rd line drawing in Fig. 2.21 by the method in [50].

Table 2.1: Times (seconds) taken by searching for internal faces (T1) and 3D reconstruction (T2) for all the line drawings (a)–(p) in Figs. 2.20 and 2.21.

	T1	T2		T1	T2		T1	T2		T1	T2
a	< 1	< 1	e	8	6	i	4	5	m	10	30
b	< 1	< 1	f	4	1	j	5	9	n	3	21
c	< 1	< 1	g	2	4	k	14	7	o	4	24
d	2	< 1	h	2	3	l	12	12	p	8	45

previous methods. However, it is still unable to handle the complex line drawings in Fig. 2.20 and Fig. 2.21. For example, for the line drawing (f) and (k), it generates results that are not what we want to have, as shown in Fig. 2.22.

The computational time of our algorithm varies with different drawings, depending on their complexity. The main computation cost comes from the search of internal faces and the 3D reconstruction. Table 2.1 lists the times taken by these two parts for all the line drawings in Figs. 2.20 and 2.21.

From these experimental results, we can see that some are not perfect, which is partly due to the inaccurate sketches of the line drawings. The beautification of the reconstructed objects and the improvement of the computational efficiency of the algorithm are our future work.

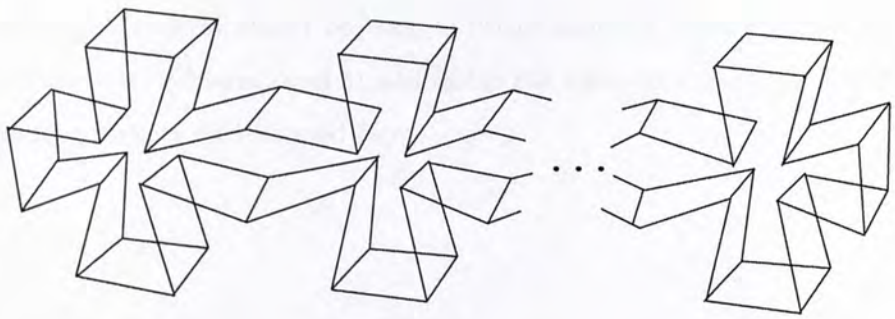


Fig. 2.23: A complex line drawing without internal faces.

Our current approach can handle complex line drawings which contain one or more internal faces. It is worth to mention that there do exist some special cases when complex line drawings may not have internal faces. An example is given in Fig. 2.23, in which many crosses are stuck together but no internal faces are formed. In these cases, our approach is degenerated into a direct 3D reconstruction method without the divide-and-conquer strategy. However, in most cases, humans are more likely to create line drawings with internal faces and hence a divide-and-conquer approach can usually be effective. Better dealing with those complex line drawings without internal faces is also our future work.

2.6 Summary

In this chapter, we have proposed a novel divide-and-conquer algorithm for 3D complex manifold reconstruction from single line drawings. Our strategy is to 1) identify the internal face in the line drawing, 2) separate a complex line drawing into simpler ones along its internal faces, 3) reconstruct the 3D shapes from these simpler line drawings, and 4) combine the shapes into a complete object. The experiments show that our approach can handle more complex objects than the solid objects appearing in the previous related papers.

Future work includes 1) the improvement of the computational efficiency of the algorithm, 2) the beautification of the reconstructed objects, 3) the extension of the

work to handle general planar objects, 4) better handling those complex line drawings without internal faces, and 5) addressing the reconstruction from line drawings representing objects with curved faces.

Chapter 3

A Vision-Based Sketching System for 3D Object Design

3.1 Introduction

3D object design has many applications including flexible 3D sketch input in CAD, computer game, webpage content design, image based object modeling, and 3D object retrieval. Despite great progress of 3D modeling in current computer-aided design (CAD) tools, creating 3D objects using these tools is still a tedious job since they require users to work on a 2D drawing plane. Design in virtual 3D environments enables users to draw objects in 3D space, but this method has the drawbacks that there are awkward devices worn by the user and the virtual environments are expensive.

In this chapter, we propose a novel vision-based approach to 3D object design. Different from the current techniques, it works in 3D space without any devices connected to the user. Our target is to develop an inexpensive system that allows the user to design 3D objects conveniently. Our system consists of a PC, a camera, and a mirror, as shown in Fig. 3.1. In the system, the user designs 3D objects by sketching in the air the wireframes of the objects with an easily-tracked wand. A number of sketching and editing operations are developed to facilitate object design. In real time, the 3D positions of the strokes of the wand are captured, and

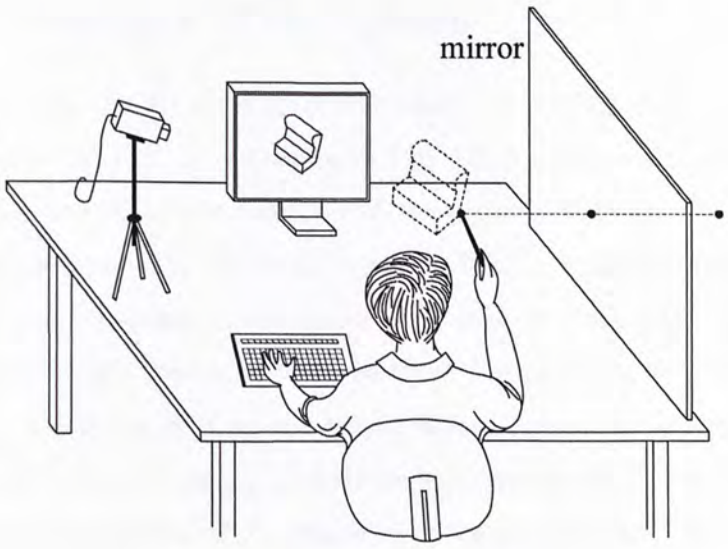


Fig. 3.1: The sketching system.

the wireframes and surfaces being developed are displayed on the PC screen to guide the user to draw more and more complex objects.

The system provides a new way of 3D object design. It requires no special equipments and is easy to set up and use. Its applications include flexible 3D sketch input in CAD, game, education, and webpage content design, generation of 3D objects from 2D images, and a user-friendly query interface for 3D object retrieval.

3.2 The Sketching System

As shown in Fig. 3.1, our system consists of a video camera, a mirror, and a PC only. The user draws an object with a wand in the 3D free space. The tip of the wand is colored so that it is easy to track. The basic idea of 3D design in this system is that a 3D wireframe of an object is obtained by tracking the movement of the wand, and then an automatic filling-in process generates a surface from the wireframe. We propose this system based on the observation that a designer thinks not in terms of surfaces, but rather in terms of the feature curves of an object which construct the wireframe. The whole flow chart of our system is shown in Fig. 3.2.

3.3 3D Geometry of the System

Being able to find the 3D position of the wand[1] is the first step. In the system, the world frame (X, Y, Z) is defined as in Fig. 3.3, where the XY plane is parallel to the image plane xy of the camera and the distance between the origin $\mathbf{O}$ and the image plane is equal to the focal length f. With a simple calibration, the YZ plane can be set orthogonal to the mirror. The angle θ between the Z axis and the mirror is less than 90 degrees so that the tip of the wand $\mathbf{P}_1 = (X_1, Y_1, Z_1)^T$ and its image $\mathbf{P}_2 = (X_2, Y_2, Z_2)^T$ in the mirror always project to two different points $\mathbf{p}_1 = (x_1, y_1, f)^T$ and $\mathbf{p}_2 = (x_2, y_2, f)^T$ on the image plane. To find the 3D coordinate of $\mathbf{P}_1$, we need to know θ and Z_0, where Z_0 is the distance from $\mathbf{O}$ to $\mathbf{A}$ and $\mathbf{A}$ is the intersection of the Z axis and the mirror. The two parameters θ and Z_0 can be obtained by the calibration scheme discussed in Section 3.4.2.

The 3D position $\mathbf{P}_1 = (X_1, Y_1, Z_1)^T$ can be determined from the known θ, Z_0, $\mathbf{p}_1$, $\mathbf{p}_2$, and the geometrical relationship shown in Fig. 3.3. The formula for calculating $\mathbf{P}_1$ is derived in Section 3.4.1.

Our system is able to determine the 3D positions of the wand with sufficient accuracy. We have also tested the traditional stereo method using two cameras to find the depth of a spatial point. From our experiments, we have found that the

[1]More exactly, the position of the tip of the wand.

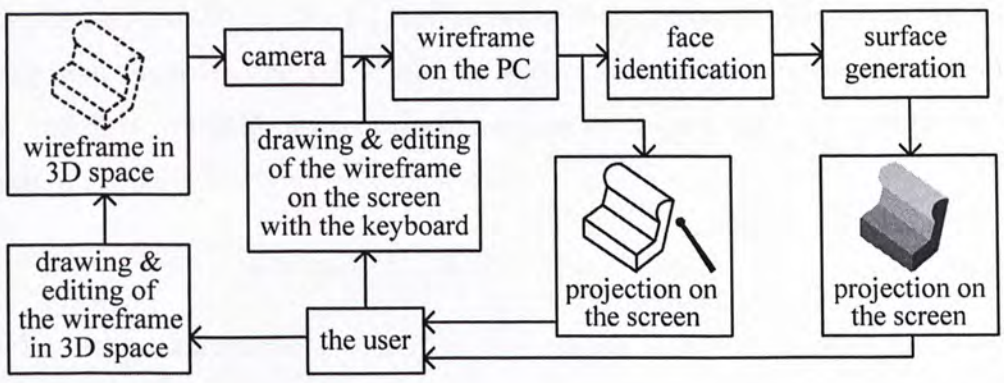


Fig. 3.2: Flow chart of 3D object design in the sketching system.

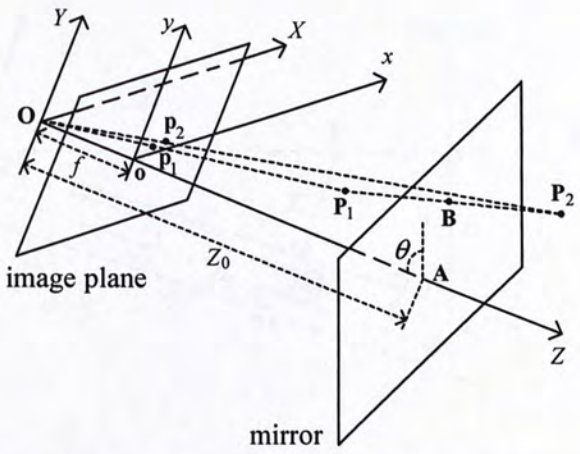


Fig. 3.3: Geometry of the system.

new method has the following advantages: (a) easier to calibrate, (b) less time to track the wand due to only one video sequence to handle, and (c) larger 3D working space for object drawing if the volume to set up the two systems are the same. The last advantage comes from the fact that if the traditional stereo method is used, the tip of the wand must appear in both image sequences of the cameras, which limits the 3D drawing space.

3.3.1 Locating the Wand

Locating the 3D position $\mathbf{P}_1$ of the wand is the first step for the system to work. We can represent $\mathbf{P}_1 = (X_1, Y_1, Z_1)^T$ in terms of the points $\mathbf{p}_1$ and $\mathbf{p}_2$ on the image plane, and the parameters f, θ, and Z_0. First, from the geometrical relationship in Fig. 3.3, it is straightforward to relate the spatial positions with the positions in the image plane by

$$X_i = x_i Z_i / f, \quad Y_i = y_i Z_i / f, \quad i = 1, 2. \tag{3.1}$$

We now consider the geometry of the system on the YZ plane as shown in Fig. 3.4, where $\mathbf{P}$ and $\mathbf{P}'$ are the projections of $\mathbf{P}_1$ and $\mathbf{P}_2$ onto the YZ plane, respectively. Let $\mathbf{B}$ be the midpoint of $\mathbf{PP}'$, and the three lines $\mathbf{PQ}$, $\mathbf{P}'\mathbf{Q}'$, and $\mathbf{BC}$

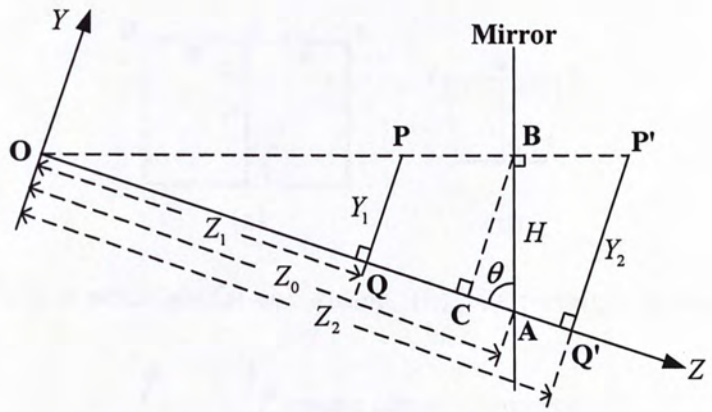


Fig. 3.4: Geometry of the system on the YZ plane.

be perpendicular to the axis **OZ**. Then $|\mathbf{PQ}|+|\mathbf{P'Q'}| = 2|\mathbf{BC}|$ and $|\mathbf{OQ}|+|\mathbf{OQ'}| = 2|\mathbf{OC}|$. Hence, we have

$$\begin{aligned} Y_2 &= |\mathbf{P'Q'}| = 2|\mathbf{BC}| - |\mathbf{PQ}| \\ &= 2H\sin\theta - Y_1 \\ &= 2H\sin\theta - y_1 Z_1/f, \end{aligned} \tag{3.2}$$

$$\begin{aligned} Z_2 &= |\mathbf{OQ'}| = 2|\mathbf{OC}| - |\mathbf{OQ}| \\ &= 2(Z_0 - H\cos\theta) - Z_1. \end{aligned} \tag{3.3}$$

On the other hand,

$$\begin{aligned} H &= |\mathbf{AB}| = |\mathbf{PQ}|\sin\theta + |\mathbf{AQ}|\cos\theta \\ &= Y_1\sin\theta - (Z_1 - Z_0)\cos\theta \\ &= y_1 Z_1 \sin\theta/f - (Z_1 - Z_0)\cos\theta. \end{aligned} \tag{3.4}$$

From (3.1), (3.2), (3.3), and (3.4), we have,

$$\begin{aligned} \frac{y_2}{f} &= \frac{Y_2}{Z_2} = \frac{2H\sin\theta - y_1 Z_1/f}{2(Z_0 - H\cos\theta) - Z_1} \\ &= \frac{2y_1 Z_1 \sin^2\theta - (Z_1 - Z_0)f\sin 2\theta - y_1 Z_1}{2Z_0 f\sin^2\theta - y_1 Z_1 \sin 2\theta + Z_1 f\cos 2\theta}. \end{aligned} \tag{3.5}$$

Finally, Z_1 is obtained from (3.5) as

$$Z_1 = \frac{2fZ_0\sin\theta(y_2\sin\theta - f\cos\theta)}{(y_1 y_2 - f^2)\sin 2\theta - f(y_1 + y_2)\cos 2\theta}. \tag{3.6}$$

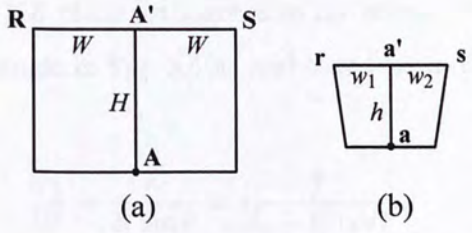


Fig. 3.5: (a) A rectangle for calibration. (b) The rectangle in the image.

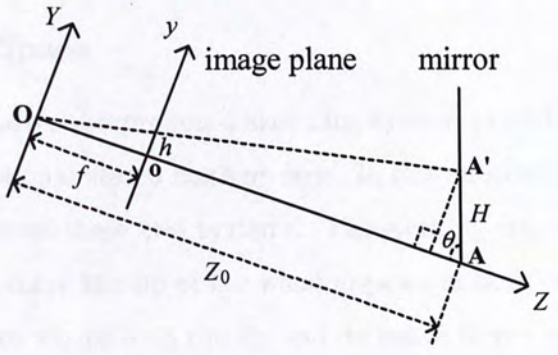


Fig. 3.6: Geometry of the YZ plane in calibration.

Given (3.1), we immediately have the other two dimensions of $\mathbf{P_1}$: $X_1 = x_1 Z_1/f$ and $Y_1 = y_1 Z_1/f$. The position of the wand $\mathbf{P_1}$ in the 3D space is hence determined if f, θ, and Z_0 are known.

3.3.2 Calibration

The calibration process is to find the parameters θ and Z_0. It is reasonable to assume that the Z axis in Fig. 3.3 is orthogonal to the image plane and passes through the center **o** of the image displayed on the screen. The focal length f can be known from the camera and is fixed in the system. To calibrate the system, we print out a rectangle (Fig. 3.5(a)) on a white page and place this page on the central part of the mirror with the side **RS** approximately parallel to the ground. This rectangle is captured by the camera and displayed on the screen as shown in Fig. 3.5(b). Then we adjust the position of the camera so that the point **a** is coincident with the image center **o**, the side **rs** is horizontal, and $w_1 = w_2$ in the image. This simple

adjustment makes the YZ plane orthogonal to the mirror. With the known lengths of the sides of the rectangle in Fig. 3.5(a) and h and w_1 in the image, we can find θ and Z_0 from

$$\frac{w_1}{W} = \frac{h}{H \sin\theta} = \frac{f}{Z_0 - H\cos\theta}, \tag{3.7}$$

which is derived from the geometry shown in Fig. 3.6.

3.3.3 Working Space

As we mentioned above, the proposed sketching system provides a larger working space than the traditional stereo method does. In this subsection, we compare the working spaces between these two systems. The working space in the traditional system is the space where the tip of the wand appears in both cameras. In the new system, it is the space where both the tip and its image in the mirror appear in the camera.

The calculation is based on a fixed setting that the distance from the center(s) of the camera(s) to the wall or the mirror is 1 meter, and the focal lengths of the cameras are all set to 3.6mm. Besides, the camera's internal image size is $2.53 \times 3.60\text{mm}^2$. We use a Monte-Carlo method [27] to calculate the working spaces of the two systems under different settings. For the two-camera system, we adjust the distance d between the two cameras on the baseline, and the angle ϕ between the camera axis and the normal of the wall, as shown in Fig. 3.7(a). For the one-camera-plus-a-mirror system, we consider the working space with different α, the angle between the camera axis and the normal of the mirror, as shown in Fig. 3.7(b). The parameters are changed within reasonable ranges. The results are given in Table 3.1 and Table 3.2.

Theoretically, the larger the α, the larger the working space in the new system. However, a large α causes the camera to be set too high and far from the working space, which causes the tracking of the wand unreliable. In practice, we choose $\alpha = 20$ degrees. From Table 3.1 and Table 3.2, we can see that the working space

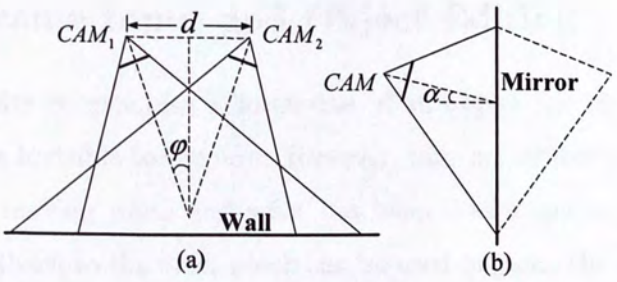


Fig. 3.7: (a) Geometry of the two-camera system. (b) Geometry of the one-camera-plus-a-mirror system.

Table 3.1: The working space of the two-camera system.

Working Space ($\times 10^3 cm^3$)		d (cm)			
		20	30	40	50
ϕ (degree)	0	133.1	90.9	54.2	25.6
	5	197.7	153.1	111.6	75.0
	10	188.1	200.7	178.2	137.3
	15	140.5	173.7	193.4	193.5
	20	86.6	127.5	161.1	184.8

of the new system is about 30% larger than the maximum working space of the two-camera system. This is due to the fact that the tip of the wand has to appear in both cameras in the traditional system, while it only needs to appear in one camera in the new system.

Table 3.2: The working space of the one-camera-plus-a-mirror system.

α (degree)	10	15	20	25
Working Space ($\times 10^3 cm^3$)	227.6	238.8	255.7	269.7

3.4 Wireframe Input and Object Editing

The main difficulty to generate a wireframe of an object lies in the fact that the strokes drawn are invisible to the user. However, they are visible to the camera, and the trace of the moving wand and what has been drawn can be displayed on the screen as the feedback to the user, which can be used to guide the sketching process.

First, we propose a solution to locating the position on an unfinished wireframe in order to continue to draw. While the user moves the wand in the space, the 3D coordinates of the tip are given on the screen. Besides, the closest point on the unfinished wireframe to the tip is computed, and a different color is shown on the screen to indicate this point. In this way, the user can find a position to draw without difficulty. When this position is found, the user presses some key to let the system know it, and then the movement of the tip is considered as a new edge of the wireframe. An illustration to the sketching input is given in Fig. 3.8.

Second, to distinguish a drawing stroke from non-drawing movement of the wand, we use the keyboard to let the system know when a stroke begins and ends. Besides, we have also developed a number of sketching and editing operations to facilitate object design, such as extrusion, moving, copying, rotation, and zooming. These 3D operations and gestures are summarized in Table 3.3. The keyboard is used to control the start and stop of a 3D gesture shown by the movement of the wand. All the operations can be done by the wand and the keyboard, without resorting to the mouse, thus allowing a continuous design by moving the wand with one hand and hitting the keyboard with another. The system also allows the user to switch between two drawing modes: the curve mode and the straight-line mode. In the curve-mode, smooth Bezier curves are generated to fit the path of wand movement when forming the wireframe. On the other hand, in the straight-line mode, the path information is discarded and only the start and the end positions of each stroke is used to generate straight lines.

Compared with traditional sketch-based editing operations on a 2D plane, many 3D operations have their advantages. For example, if we want to draw a duct by the

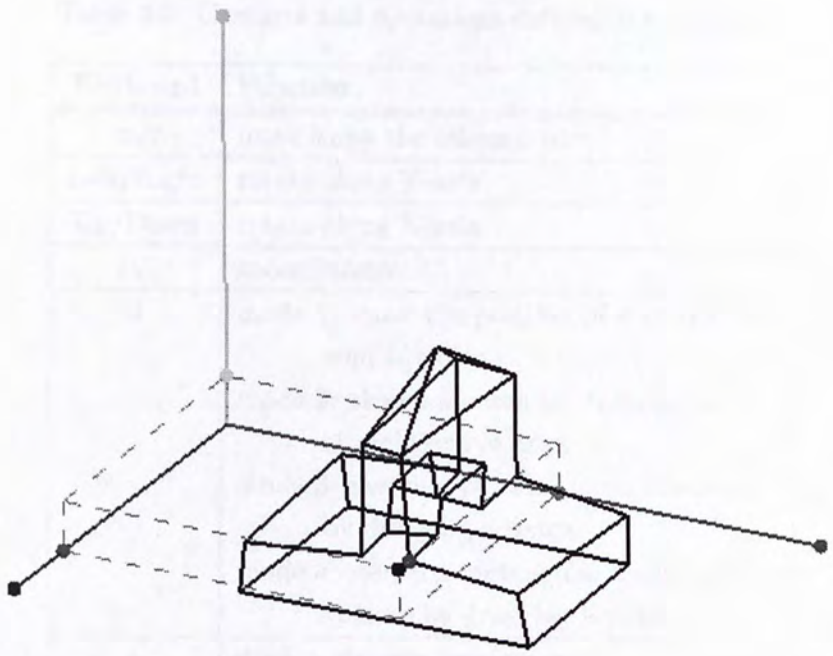


Fig. 3.8: The sketching input. The black dot shows the position of the tip of the wand. The red, blue, and green dots indicate the x, y, and z-coordinates of the tip, respectively. The closest point on the unfinished wireframe to the tip is shown by the gray dot.

extrusion of a closed circle along an arbitrary open curve (see Fig. 3.9), a 2D system will encounter two problems: (a) whether the closed curve is a circle or ellipse and its orientation in 3D space are unclear; (b) the 3D trail of the open curve is impossible to determine. However, these problems do not exist in our system.

3.5 Surface Generation

When we obtain the wireframe of an object in the 3D space by using the sketching scheme described in the previous section, the next step is to generate the 3D surface from the wireframe to finally reconstruct the object. Surface generation can be divided into two steps. The first is to identify all the faces, i.e., the circuits in the wireframe which represent patches constituting the whole surface, and the second

Table 3.3: Gestures and operations defined in the system.

Keyboard	Function
m/r	move/copy the selected part
Left/Right	rotate along Y-axis
Up/Down	rotate along X-axis
+/-	zoom in/out
d	mode 1: move the position of a vertex or a control point mode 2: sketch a patch by dragging a straight/curve edge mode 3: sketch a pyramid/cone structure by dragging a patch mode 4: sketch a rectangular/cylindrical volume by dragging a patch
a	draw a straight line/curve
c/e	mode 1: draw a circle/ellipse mode 2: draw a body of revolution from a straight/curve edge
q/w	adjust the scaling of the selected part
z/x	adjust the curvature of curved strokes
1-4	change editing modes
F1	switch between the curve mode and the straight-line mode

step is to generate these patches, either planar or curved, from their boundaries.

3.5.1 Face Identification

Given a wireframe, before filling in it with surface patches, we have to identify the circuits that represent these patches (faces). Since the wireframe may represent a manifold or non-manifold solid, a sheet of surface, or the combination of them, with or without holes, identifying the faces is not a trivial problem due to the combinatorial explosion in the number of circuits in the wireframe. To solve this problem,

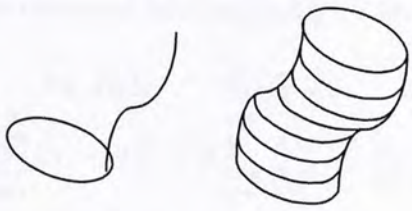


Fig. 3.9: Extrusion of the circle along the curve.

we use the algorithm proposed in [53] to detect the faces of a wireframe. In our interactive system, the user can also select the edges of a face for face identification manually, which can help fix wrongly detected faces by the algorithm occasionally.

3.5.2 Planar Surface Generation

An object may have many planar faces. In the system, a straight line replaces a stroke in the straight-line mode. It is reasonable to consider that a face is planar if all its edges are straight lines. However, for a planar face with more than three vertices, it is not likely for all the vertices to be located exactly on a plane in 3D space due to the inaccuracy of the measurement and the input during the sketching process. Filling in these circuits with triangular patches will make the object distorted. In order to solve this problem, we propose an automatic line drawing correction algorithm to deal with this problem.

After face identification from a (partial) wireframe, we know the circuits representing planar faces. From the vertices of these circuits, a fitting algorithm is used to find a set of planes that best fit these planar circuits. We represent a plane passing through face j by its normal vector $\mathbf{f}_j = (a_j, b_j, c_j)^T$ and a scale d_j. Then, any vertex $\mathbf{v} = (x, y, z)^T$ on this plane satisfies the linear equation: $a_j x + b_j y + c_j z - d_j = 0$ or $\mathbf{v}^T \mathbf{f}_j = d_j$.

We hope that for each identified face, the corrected positions of its vertices should be as close to the fitting plane as possible. Besides, for each vertex, its corrected position should not deviate too much from its initial position. Let $\mathcal{V}_j$ be the set of

the vertices of face j. The objective function to be minimized is defined as follows:

$$Q(\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_N, \mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_M, d_1, d_2, \cdots, d_M) = \sum_{i=1}^{N} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \beta \sum_{j=1}^{M} \sum_{i \in \mathcal{V}_j} \frac{\|\mathbf{v}_i^T \mathbf{f}_j - d_j\|^2}{\|\mathbf{f}_j\|^2}, \tag{3.8}$$

where $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_N$ are the corrected positions of N vertices; $(\mathbf{f}_1, d_1), (\mathbf{f}_2, d_2), \cdots, (\mathbf{f}_M, d_M)$ are the parameters of M planar faces; $\mathbf{v}_1', \mathbf{v}_2', \cdots, \mathbf{v}_N'$ are the positions of the N vertices in the original sketching; β is a weighting factor. The goal of the optimization is to find the corrected positions $\mathbf{v}_i$, $i = 1, 2, \cdots, N$, and the fitting planes $(\mathbf{f}_j, d_j)$, $j = 1, 2, \cdots, M$, such that Q is minimized.

We solve this optimization problem in an iterative way. Let the set of $\mathbf{v}_i$, $i = 1, 2, \cdots, N$, the set of $\mathbf{f}_j$, $j = 1, 2, \cdots, M$, and the set of d_j, $j = 1, 2, \cdots, M$ be $V = \{\mathbf{v}_i\}_{i=1}^N$, $F = \{\mathbf{f}_j\}_{j=1}^M$, and $D = \{d_j\}_{j=1}^M$, respectively. Also let $V^n = \{\mathbf{v}_i^n\}_{i=1}^N$, $F^n = \{\mathbf{f}_j^n\}_{j=1}^M$, and $D^n = \{d_j^n\}_{j=1}^M$ be the optimization results after the nth iteration. The optimization problem is divided into two iterative minimization steps, and a closed-form solution can be achieved in each step.

Step 1: Face fitting.

$$(F^{n+1}, D^{n+1}) = \arg\min_{F,D} Q(V^n, F, D). \tag{3.9}$$

Step 2: Vertex correction.

$$V^0 = \{\mathbf{v}_i'\}_{i=1}^N, \tag{3.10}$$

$$V^{n+1} = \arg\min_{V} Q(V, F^{n+1}, D^{n+1}). \tag{3.11}$$

In Step 1, we do the plane fitting on all the identified planar faces using the updated positions of the vertices obtained in the previous iteration. The optimal fitting is obtained as follows. First, by solving $\frac{\partial Q}{\partial d_j} = 0$, we have

$$d_j = \frac{1}{|\mathcal{V}_j|} \sum_{i \in \mathcal{V}_j} \mathbf{v}_i{}^T \mathbf{f}_j, \quad j = 1, 2, \cdots, M. \tag{3.12}$$

Substituting (3.12) into (3.8), after some algebraic manipulation, we can transform the problem in (3.9) into the problem of minimizing the Rayleigh quotient $\frac{\mathbf{f}_j^T \mathbf{S}_j \mathbf{f}_j}{\mathbf{f}_j^T \mathbf{f}_j}$

with respect to each $\mathbf{f}_j$, $j = 1, 2, \cdots, M$, where $\mathbf{S}_j = \frac{1}{|\mathcal{V}_j|}\sum_{i\in\mathcal{V}_j}(\mathbf{v}_i - \overline{\mathbf{v}}_j)(\mathbf{v}_i - \overline{\mathbf{v}}_j)^T$ is the covariance matrix, and $\overline{\mathbf{v}}_j = \frac{1}{|\mathcal{V}_j|}\sum_{i\in\mathcal{V}_j}\mathbf{v}_i$. Furthermore, minimizing $\frac{\mathbf{f}_j^T\mathbf{S}_j\mathbf{f}_j}{\mathbf{f}_j^T\mathbf{f}_j}$ can be reduced to the following eigen-problem:

$$\mathbf{S}_j\mathbf{f}_j = \lambda_{j,min}\mathbf{f}_j, \quad j = 1, 2, \cdots, M, \tag{3.13}$$

with $\mathbf{f}_j$ being the eigen vector corresponding to the minimum eigen value $\lambda_{j,min}$ of $\mathbf{S}_j$. From (3.12) and (3.13), we can obtain the closed-form solution $\mathbf{f}_j^{n+1}$ and d_j^{n+1}, $j = 1, 2, \cdots, M$, in terms of $\mathbf{v}_i^n$, $i = 1, 2, \cdots, N$.

Step 2 is done by minimizing Q, given the fitting planes obtained in Step 1. From $\frac{\partial Q}{\partial \mathbf{v}_i} = \mathbf{0}$, $i = 1, 2, \cdots, N$, we have

$$\frac{\partial Q}{\partial \mathbf{v}_i} = 2(\mathbf{v}_i - \mathbf{v}_i') + 2\beta\sum_{j\in\mathcal{F}_i}\frac{(\mathbf{v}_i^T\mathbf{f}_j - d_j)\mathbf{f}_j}{\|\mathbf{f}_j\|^2} = \mathbf{0}, \tag{3.14}$$

where $\mathcal{F}_i$ is the set of the faces containing vertex i.

The closed form solution to (3.14) results in the following equation for computing $\mathbf{v}_i^{n+1}$:

$$\mathbf{v}_i^{n+1} = (\mathbf{R}^{n+1})^{-1}\hat{\mathbf{v}}_i^{n+1}, \quad i = 1, 2, \cdots, N, \tag{3.15}$$

where

$$\mathbf{R}^{n+1} = \mathbf{I} + \beta\sum_{j\in\mathcal{F}_i}\frac{\mathbf{f}_j^{n+1}\mathbf{f}_j^{n+1^T}}{\|\mathbf{f}_j^{n+1}\|^2}, \tag{3.16}$$

$$\hat{\mathbf{v}}_i^{n+1} = \mathbf{v}_i' + \beta\sum_{j\in\mathcal{F}_i}\frac{d_j^{n+1}\mathbf{f}_j^{n+1}}{\|\mathbf{f}_j^{n+1}\|^2}. \tag{3.17}$$

Since both steps minimize the object function and have closed-form solutions, the convergence of this algorithm can always be guaranteed. Our experiments have shown that the algorithm is effective for finding the global minimum in most cases and converges quickly within several iterations.

3.5.3 Smooth Curved Surface Generation

After a curved stroke is finished in the curve mode, we use a Bezier curve to approximate it to obtain a smooth curve. When we have a (partial) wireframe with

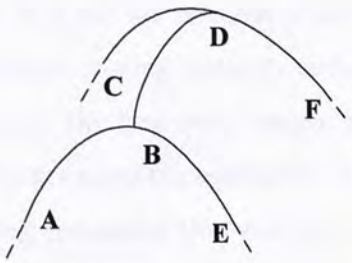


Fig. 3.10: Smoothness between two patches.

identified faces (circuits) and curves, we fill in the circuits denoting curved faces with smooth surface patches.

In our work, we use triangle meshes to generate curved surfaces. Triangle meshes can be used to represent surfaces with all boundary types, unlike parametric patches such as Bezier and Coons patches, which are mainly suitable for patches with four sides. Another advantage of using triangle meshes is that they allow more flexible local editing.

We first build an initial isotropic mesh for each circuit denoting a curved face from its boundary. The curved surface patch is then generated through optimizing the initial meshes according to the following three criteria. First, the surface should be smooth not only inside a patch, but also along the boundary between two neighboring patches, as shown in Fig. 3.10, if the curves $\widetilde{\mathbf{BD}}$, $\widetilde{\mathbf{ABE}}$, and $\widetilde{\mathbf{CDF}}$ in the wireframe are smooth. Second, the curvature of the surface should be continuous. Third, a generated patch should fit into its boundary well. With these constraints, the mesh generation can be formulated as an quadratic optimization problem of minimizing the following objective function P:

$$P(\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_K) = \lambda \sum_i \sum_{j \in \mathcal{N}(i)} \|\mathbf{u}_i - \mathbf{u}_j\|^2 + \gamma \sum_{i \in \mathcal{S} \setminus \mathcal{B}} \sum_{j \in \mathcal{N}(i) \setminus \mathcal{B}} \|\mathbf{c}_i - \mathbf{c}_j\|^2 + \sum_{i \in \mathcal{S}} \|\mathbf{u}_i - \mathbf{u}'_i\|^2, \tag{3.18}$$

where $\mathbf{u}'_i$, $\mathbf{u}_i$, and $\mathbf{c}_i$, $i = 1, 2, \cdots, K$, are the initial positions, the new positions, and the curvatures of all K mesh points, respectively; $\mathcal{N}(i)$ is the set of mesh points connected to the ith mesh point in the mesh; $\mathcal{S}$ is the set of mesh points located

on the input wireframe; $\mathcal{B}$ is the set of mesh points located on the strokes of the wireframe where the surface passing through them is not smooth; λ and γ are weighting factors. In (3.18), the first term represents the smoothness of the mesh, the second term is used to maintain the continuity of the curvature in the mesh, and the last term is the fitting constraint that requires the meshes to fit the points on the wireframe well.

One simple way to approximate the curvature in a mesh is to use the discrete graph Laplacian [80]:

$$\mathbf{c}_i = \mathbf{u}_i - \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{u}_j, \quad i \in \mathcal{S} \setminus \mathcal{B}. \tag{3.19}$$

To find the solution to minimizing P, we rewrite (3.18) in matrix form:

$$\begin{aligned} P(\mathbf{U}) = {} & \lambda \mathbf{Tr}(\mathbf{U}^T \mathbf{L} \mathbf{U}) + \gamma \mathbf{Tr}(\mathbf{C}^T \mathbf{L}' \mathbf{C}) \\ & + \mathbf{Tr}\big((\mathbf{U} - \mathbf{U}')^T \mathbf{I}_S (\mathbf{U} - \mathbf{U}')\big), \end{aligned} \tag{3.20}$$

where $\mathbf{U} = [\mathbf{u}_1^T, \mathbf{u}_2^T, \cdots, \mathbf{u}_K^T]^T$, $\mathbf{U}' = [\mathbf{u}'^T_1, \mathbf{u}'^T_2, \cdots, \mathbf{u}'^T_K]^T$, $\mathbf{C} = [\mathbf{c}_1^T, \mathbf{c}_2^T, \cdots, \mathbf{c}_K^T]^T = \mathbf{HU}$, $\mathbf{I}_S = diag[s_1, s_2, \cdots, s_K]$ (diagonal matrix) with $s_k = 1$ if $k \in \mathcal{S}$ and $s_k = 0$ if $k \notin \mathcal{S}$, $\mathbf{L} = [L_{i,j}]_{K \times K}$, $\mathbf{L}' = [L'_{i,j}]_{K \times K}$, and $\mathbf{H} = [H_{i,j}]_{K \times K}$ with

$$L_{i,j} = \begin{cases} |\mathcal{N}(i)|, & i = j, \\ -1, & i \neq j, \ j \in \mathcal{N}(i), \\ 0, & \text{otherwise}, \end{cases}$$

$$L'_{i,j} = \begin{cases} |\mathcal{N}(i) \setminus \mathcal{B}|, & i = j, \ i \notin \mathcal{B}, \\ -1, & i \neq j, \ i \notin \mathcal{B}, \ j \in \mathcal{N}(i) \setminus \mathcal{B}, \\ 0, & \text{otherwise}, \end{cases}$$

$$H_{i,j} = \begin{cases} 1, & i = j, \ i \notin \mathcal{B}, \\ -1/|\mathcal{N}(i)|, & i \neq j, \ i \notin \mathcal{B}, \ j \in \mathcal{N}(i), \\ 0, & \text{otherwise}. \end{cases}$$

The problem of minimizing $P(\mathbf{U})$ can be solved as follows:

$$\left.\frac{\partial P}{\partial \mathbf{U}}\right|_{\mathbf{U}=\mathbf{U}^*} = 2\lambda\mathbf{L}\mathbf{U} + 2\gamma\mathbf{H}^T\mathbf{L}'\mathbf{H}\mathbf{U} + 2\mathbf{I}_S(\mathbf{U} - \mathbf{U}') = \mathbf{0}. \tag{3.21}$$

Finally the closed form solution is

$$\mathbf{U}^* = \left(\mathbf{I}_S + \lambda\mathbf{L} + \gamma\mathbf{H}^T\mathbf{L}'\mathbf{H}\right)^{-1}\mathbf{I}_S\mathbf{U}', \tag{3.22}$$

which gives the positions of all the mesh points.

3.6 Experiments

In this section, we show a number of examples to demonstrate the performance of our system. Our system is implemented using Visual C++, running on a PC with 3.4 GHz Pentium IV CPU. The parameters β in (3.8), and λ and γ in (3.20) are chosen to be 10, 0.05, and 0.2, respectively. Our experiments show that the system is insensitive to the parameters; very similar results are obtained when β, λ, and γ change in $[5, 20]$, $[0.02, 0.1]$, and $[0.1, 0.5]$, respectively.

The resolution of the video sequence in our system is set to 320×240. Although the low resolution results in larger errors in tracking, our experiments show that it only slightly affects the accuracy of the sketching process when zooming operation is used. On the other hand, it greatly enhances the processing speed of the system. The wand tracking and display module works in real time at a rate of 10 frames per second. The system can track the moving of the wand at a maximum speed of about one meter per second.

A new user usually needs to take two or three hours of training to get adapted to simultaneous keyboard and wand operation in the system. In our system, strokes are preprocessed and the displayed wireframes are composed of straight lines and smooth curves. The jittering of the strokes by natural hand tremor is smoothed.

Fig. 3.11 shows a set of wireframes created with our system. For each wireframe in the first column, we give the corrected wireframes in the second column and the

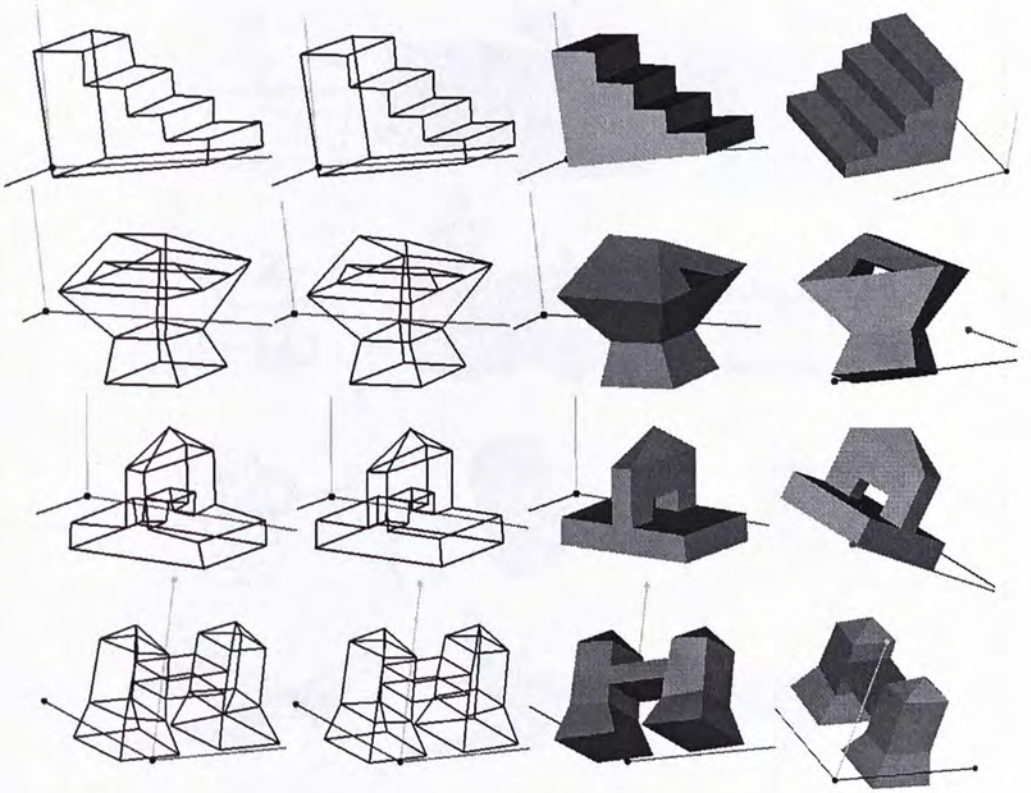


Fig. 3.11: Experimental results. The axes of the 3D coordinate system are also shown.

3D reconstruction result displayed in two views in the third and fourth columns. The results show that the correction step (Section 3.5.2) is effective and the faces of the reconstructed objects are planar.

Figs. 3.12 and 3.13 show a set of more complex objects that include strokes of both straight-lines and curves. We can see that our system can handle complex wireframes sketched in the air and generate expected 3D objects.

The time used for the face identification and the generation of planar and curved surfaces of a scene is between 3 and 48 seconds, depending on the complexity of the scene. For instance, the system takes 3, 19, and 48 seconds to reconstruct the second object in Fig. 3.11, the fourth object in Fig. 3.12, and the scene in Fig. 3.13, respectively, after the wireframes are obtained.

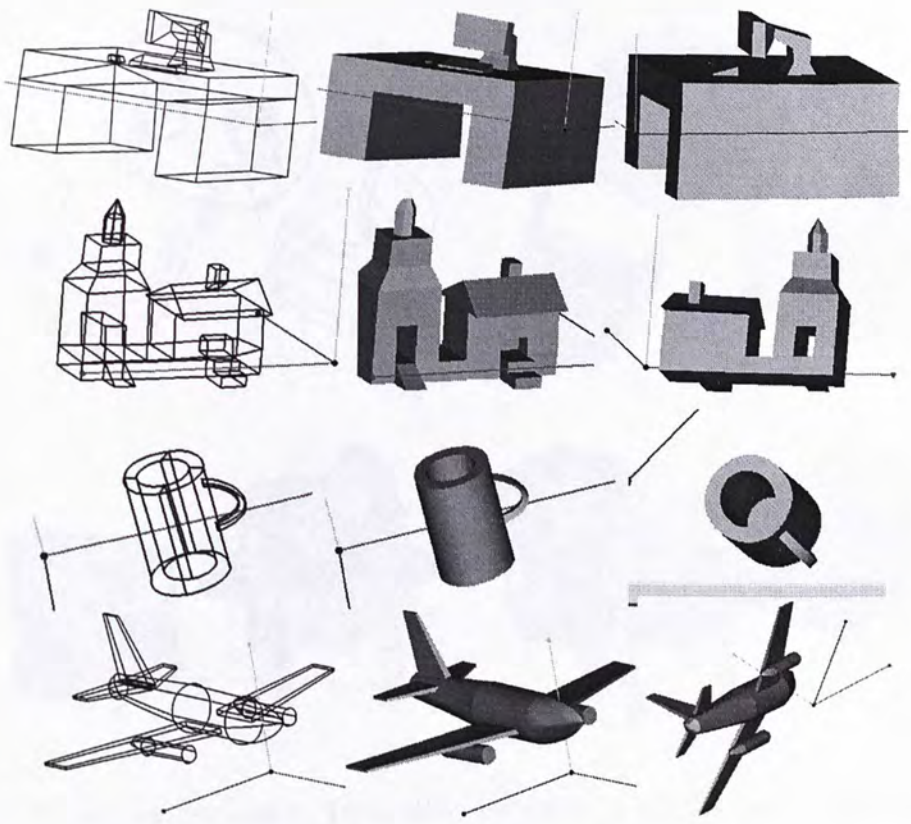


Fig. 3.12: More experimental results.

3.7 Summary

We have developed a novel 3D vision-based sketching system with a simple and inexpensive interface allowing users to sketch objects directly in the 3D space. A number of new techniques are proposed for working in this system, including input of object wireframes, gestures for editing and drawing objects, and optimization-based planar and curved surface generation. Experiments have verified its efficacy in designing 3D objects. Our system is still being improved. At the current stage, a new user usually needs to take two or three hours of training to get adapted to simultaneous keyboard and wand operation in the system. Our future work includes: 1) developing more gestures and operations to handle complex objects; 2) improving the tracking algorithm to support more accurate 3D positioning of the

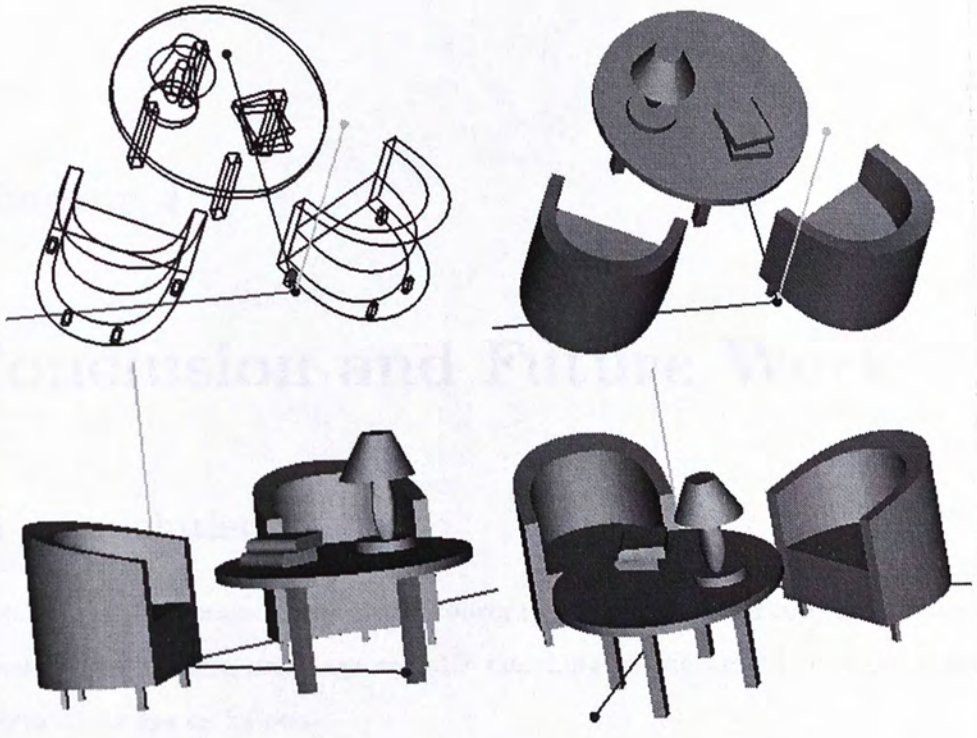


Fig. 3.13: A complex scene. Three different views of the reconstruction result are given.

wand movement; 3) conducting more user testing and making the system available to the public.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

In this thesis, we examine the issues concerning the problem of reconstructiong 3D objects from 2D line drawings and 3D sketching in the air. The main research contributions are as follows.

First, we present a novel divide-and-conquer approach for 3D complex object reconstruction from single line drawings. This research involves proposing new algorithms for identifying the internal faces from the line drawing, separating a complex line drawing into simpler ones along its internal faces, reconstructing the 3D shapes from these simpler line drawings, and combining the shapes into a complete object. Experiments show that the new method can handle solid objects which are much more complex than those appearing in the previous related papers.

Second, we have developed a novel 3D vision-based sketching system with a simple and inexpensive interface allowing users to sketch objects directly in the 3D space. Along with this system, a number of new techniques are proposed, including input of object wireframes, gestures for editing and drawing objects, and optimization-based planar and curved surface generation.

To conclude, our work on reconstruction 3D objects from 2D and 3D line drawings has generalized the previous work to many cases they cannot handel.

4.2 Future Work

With the current results, we can expect that many interesting and exciting directions still lie ahead. Some of the options are described below, and we hope one or two of them to flourish in the future work.

4.2.1 Learning-Based Line Drawing Reconstruction

All the researches on 3D reconstruction of line drawings investigate only geometric and topological clues of the line drawing and they are mainly on a basis of heuristics and human's perception. However, no research by far makes any attempt to incorporate machine learning schemes to handle this problem. Intuitively, human's ability to interpret 2D line drawings is based on experiences and a learning process. Trying to reconstruct the 3D object from 2D line drawings using learning-based methods and presenting the optimization-based reconstruction into a Bayesian framework will be our future work.

4.2.2 New Query Interface for 3D Object Retrieval

Most work done in the area of 3D object retrieval is mainly concentrated on finding 3D shape descriptors and defining similarity measurement. A detailed survey is available in [79]. 3D shape descriptors are computational shape representations (e.g., a feature vector) with which shape matching is performed. They are expected to have good properties of shape discriminating, transform invariance, computationally efficiency, robustness against noise, etc. Current work in 3D object retrieval [28], [61], [88], [92] mainly uses existing models as the queries for 3D retrieval. Recently, a few researchers [7], [11], [44] start to take into account the user's influence on the 3D object retrieval by means of introducing relevance feedback. Our next work will be focus on improving these methods by applying the techniques in Chapter 2 and providing more comfortable and straightforward query tools to the retrieval.

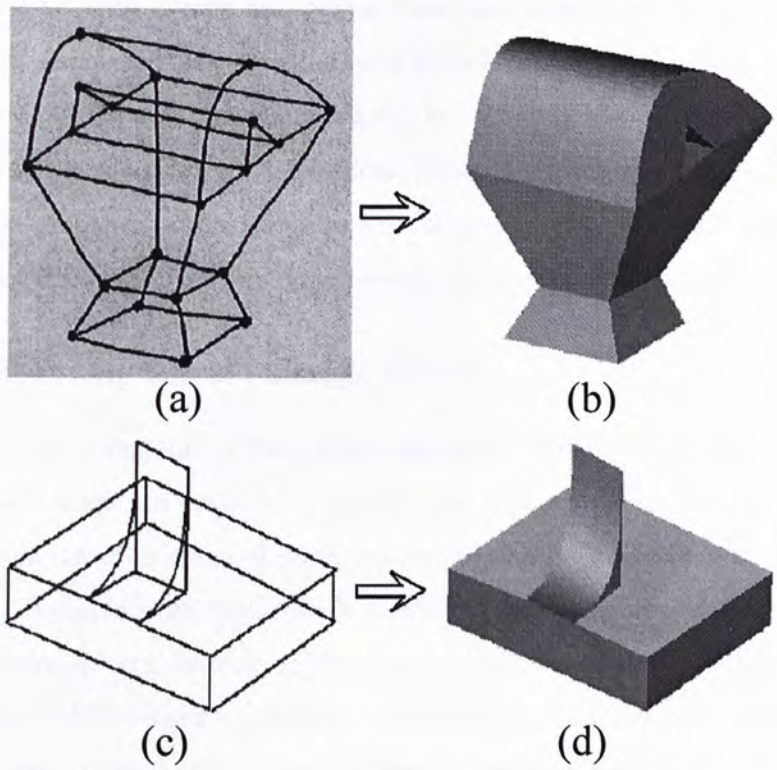


Fig. 4.1: Examples of reconstructing curved objects from line drawings.

4.2.3 Curved Object Reconstruction

Up to now, very little previous work has been done to reconstruct curved objects from line drawings (see Fig. 4.1). Although some methods [11], [86] have been proposed to handle line drawings representing curved objects, they require considerable interactions from users and only deal with objects with special geometrical traits, e.g., symmetrical objects. To the best knowledge of us, no method has been published to tackle curved objects with more complex and general geometrical structure.

Reconstruction of curved objects is a much harder problem. Three non-collinear points determine a plane, but a curved surface often has much more degrees of freedom. Therefore, the reconstruction of curved objects owns a higher underconstrained nature. Now, we are trying to investigate the problem of reconstructing

3D objects with both curved and planar faces and propose an algorithm for automatically reconstructing 3D curved objects from 2D line drawings. We will focus on labeling curved faces and planar faces in the line drawing and proposing several new regularities to characterize curved objects. Combined with the divide-and-conquer approach we proposed, curved objects with considerable complexity might be handled. Part of the work has been done recently and submitted to ECCV [91].

4.2.4 Improving the 3D Sketch System

The 3D sketching system we have implemented in Chapter 3 is still a prototype at the current stage. In order to make the system available to the public, further refinement on the interface and improvement on system controllability need to be carried out. Detailed work will include developing more gestures and operations to handle complex objects, improving the tracking algorithm to support more accurate 3D positioning of the wand movement, conducting more user testing. For example, we will consider incorporating the eye-tracking scheme into the system by introducing another web-cam. By tracking the eyes of the user, their spatial positions and relative distance from the screen can be calculated. The screen display can thus properly react to head and body movement to create the illusion of depth and space. This modification will make the 3D object that the user is drawing more intuitive and thus greatly enhance the feedback of the system.

4.2.5 Other Directions

Other research topics include extracting line drawings from images, designing faster and more robust optimization algorithms to perform the optimization, reconstruction under the perspective projection, etc.

Bibliography

[1] *Aims and Scope.* 3rd Eurographics Workshop on Sketch-Based Interfaces and Modeling, 2006.

[2] S. Ablameyko, V. Bereishik, A. Gorelik, and S. Medvedev. 3D object reconstruction from engineering drawing projections. *Computer & Control Engineering Journal*, 10(6):277–284, 1999.

[3] S.C. Agarwal and J.W.N. Waggenspack. Decomposition method for extracting face topologies from wireframe models. *Computer-Aided Design*, 24(3):123–140, 1992.

[4] A. Alexe, L. Barthe, M.P. Cani, and V. Gaildrat. Shape modeling by sketching using convolution surfaces. *Proc. Pacific Graphics*, 2005.

[5] R.D. Amicis, F. Bruno, A. Stork, and M.L. Luchi. The eraser pen: a new interaction paradigm for curve sketching in 3D. *Proc. 7th Int' l Design Conference*, 1:465–470, 2002.

[6] M. A. Armstrong. *Basic Topology.* Springer, 1983.

[7] I. Atmosukarto, W. K. Leow, and Z. Huang. Feature combinatin and relevance feedback for 3d model retrieval. *IEEE Proc. Multimedia Modelling*, pages 334–339, 2005.

[8] S. Bagali and J. Waggenspack. A shortest path approach to wireframe to solid model conversion. *Proc. 3rd Symp. Solid Modeling and Application*, pages 339–349, 1995.

[9] E. Brown and P. Wang. Three-dimensional object recovery from two-dimensional images: a new approach. *SPIE*, 2904:138–147, 1996.

[10] L. Cao, J. Liu, and X. Tang. 3D object reconstruction from a single 2D line drawing without hidden lines. *IEEE Proc. Int'l Conf. Computer Vision (ICCV)*, 1:272–277, 2005.

[11] L. Cao, J. Liu, and X. Tang. 3D object retrieval using 2D line drawing and graph based relevance feedback. *Proc. ACM Int'l Conf. Multimedia*, pages 105–108, 2006.

[12] L. Cao, J. Liu, and X. Tang. What the back of the object looks like: 3D reconstruction from line drawings without hidden lines. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(3):507–517, 2008.

[13] Y. Chen, J. Liu, and X. Tang. A divide-and-conquer approach to 3D object reconstruction from line drawings. *IEEE Proc. Int'l Conf. Computer Vision (ICCV)*, 2007.

[14] Y. Chen, J. Liu, and X. Tang. Sketching in the air: A vision-based system for 3d object design. *IEEE Proc. Computer Vision and Pattern Recognition (CVPR)*, 2008.

[15] M. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.

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

[17] P. Company, A. Piquer, M. Contero, and F. Naya. A survey on geometrical reconstruction as a core technology to sketch-based modeling. *Computer & Graphics*, 29(6):892–904, 2005.

[18] M.C. Cooper. The interpretations of line drawings with contrast failure and shadows. *Int'l Journal of Computer Vision*, 43(2):75–97, 2001.

[19] M.C. Cooper. Wireframe projections: Physicl realisability of curved objects and unambiguous reconstruction of simple polyhedra. *Int'l Journal of Computer Vision*, 64(1):69–88, 2005.

[20] M.C. Cooper. Constraints between distant lines in the labelling of line drawings of polyhedral scenes. *Int'l Journal of Computer Vision*, 73(2):195–212, 2007.

[21] M.C. Cooper. A rich discrete labeling scheme for line drawings of curved objects. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(4):741–745, 2008.

[22] SolidWorks Corporation. *SolidWorks*. http://www.solidworks.com/.

[23] S. Courter and J. Brewer. Automated conversion of curvilinear wire-frame models to surface boundary models: A topologically approach. *Computer Graphics*, 20(4):171–178, 1986.

[24] P. Debevec, C. Yaylor, and J. Malik. Modeling and rendering architecture from photograph: a hybrid geometry and image-based approach. *SIGGRAPH'96*, pages 11–20, 1996.

[25] M.F. Deering. The holosketch vr sketching system. *Communications of the ACM*, 39(5):54–61, 1996.

[26] L. Eggli, C.Y. Hsu, B.D. Bruederlin, and G. Elber. Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29(2):101–112, 1997.

[27] M. Evans and T. Swartz. *Approximating integrals via Monte Carlo and deterministic methods.* Oxford University Press, 2000.

[28] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3d models. *ACM Transactions on Graphics*, 22(1):83–105, 2003.

[29] T. Grossman, R. Balakrishnan, and K. Singh. An interface for creating and manipulating curves using a high degree-of-freedom curve input device. *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, pages 185–192, 2003.

[30] R. Haralick and L. Shapira. The consistent labeling problem: Part 1. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1(2):173–184, 1979.

[31] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. *ACM Proc. SIGGRAPH'97*, pages 225–232, 1997.

[32] D. Huffman. Impossible objects as nonsense sentences. *Machine Intelligence*, 6:295–323, 1971.

[33] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. *Proc. SIGGRAPH*, pages 406–416, 1999.

[34] Autodesk Inc. *AutoCAD.* http://www.autodesk.com/.

[35] T. Kanade. Recovery of the three-dimensional shape of an object from a single-view. *Artificial Intelligence*, 17:409–460, 1981.

[36] H.W. Kang, S.H. Pyo, K. Anjyo, and S.Y. Shin. Tour into the picture using a vanishing line and its extension to panoramic images. *Proc. EuroGraphics*, 20(3), 2001.

[37] O.A. Karpenko and J.F. Hughes. Smoothsketch: 3D free-form shapes from complex sketches. *Proc. SIGGRAPH*, pages 589–598, 2006.

[38] D.F. Keefe, D.A. Feliz, T. Moscovich, D.H. Laidlaw, and J.J. LaViola. Cavepainting: a fully immersive 3D artistic medium and interactive experience. *Symposium on Interactive 3D Graphics*, pages 85–93, 2001.

[39] M. Kuo. Reconstruction of quadric surface solid from three-view engineering drawings. *Computer-Aided Design*, 30(7):517–527, 1998.

[40] A. Kushal, G. Chanda, and K. Shrivastava. Multilevel modelling and rendering of architectural scenes. *Proc. EuroGraphics*, 2003.

[41] D. E. LaCourse. *Handbook of Solid Modeling.* New York: McGraw-Hill, 1995.

[42] N. Langrana, Y. Chen, and A. Das. Feature identification from vectorized mechanical drawings. *Computer Vision and Image Understanding*, 68(2):127–145, 1997.

[43] Y. Leclerc and M. Fischler. An optimization-based approach to the interpretation of single line drawings as 3D wire frames. *Int'l Journal of Computer Vision*, 9(2):113–136, 1992.

[44] G. Leifman, R. Meir, and A. Tal. Relevance feedback for 3d shape retrieval. *Israel Korea Conf. Geometric Modeling and Computer Graphics*, pages 15–19, 2004.

[45] R. Lequette. Automatic construction of curvilinear solid from wireframe views. *Computer-Aided Design*, 20(4):171–180, 1988.

[46] H. Li. nD polyhedra scene reconstruction from single 2D line drawing by local propagation. *LNAI*, 3763:169–197, 2006.

[47] H. Li, Q. Wang, L. Zhao, Y. Chen, and L. Huang. nD object representation and detection from simple 2D line drawing. *LNAI*, 3518:363–382, 2005.

[48] D. Liebowitz, A. Criminisi, and A. Zisserman. Creating architectural models from images. *Proc. EuroGraphics*, 18:39–50, 1999.

[49] H. Lipson and M. Shpitalni. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design*, 28(8):651–663, 1996.

[50] J. Liu, L. Cao, Z. Li, and X. Tang. Plane-based optimization for 3D object reconstruction from single line drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(2):315–327, 2008.

[51] J. Liu and Y. Lee. A graph-based method for face identification from a single 2D line drawing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(10):1106–1119, 2001.

[52] J. Liu, Y. Lee, and W. K. Cham. Identifying faces in a 2D line drawing representing a manifold object. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(12):1579–1593, 2002.

[53] J. Liu and X. Tang. Evolutionary search for faces from line drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(6):861–872, 2005.

[54] D. Lysak. Interpretation of engineering drawings of polyhedral and nonpolyhedral objects from orthographic projections. *PhD Thesis, Dept. of Electrical & Computer Engineering, The Pennsylvania State University*, 1991.

[55] A.K. Mackworth. Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4(2):121–139, 1973.

[56] W. Makela, M. Reunanen, and T. Takala. Possibilities and limitations of immersive free-hand expression: a case study with professional artists. *Proc. ACM Multimedia*, pages 504–507, 2004.

[57] J. Malik. Interpreting line drawings of curved objects. *Int'l Journal of Computer Vision*, 1(1):73–103, 1987.

[58] T. Marill. Emulating the human interpretation of line-drawings as three-dimensional objects. *Int'l Journal of Computer Vision*, 6(2):147–161, 1991.

[59] P. Min, J. Chen, and T. Funkhouser. A 2D sketch interface for a 3D model search engine. *SIGGRAPH'02, Technical Sketches*, page 138, 2002.

[60] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3d curves. *Proc. SIGGRAPH*, (41), 2007.

[61] S. Ortiz. 3D searching starts to take shape. *Computer*, 37(8):24–26, 2004.

[62] J.P. Pereira, J.A. Jorge, V.A. Branco, N.F. Silva, T.D. Cardoso, and F.N. Ferreira. Cascading recognizers for ambiguous calligraphic interaction. *Proc. EuroGraphics Workshop on Sketch-Based Interfaces and Modeling*, pages 63–72, 2004.

[63] C. Pinet. Virtual reality: heir apparent to cad. *The Journal of Computer-Aided Environmental Design and Education*, 2(1), 1996.

[64] A. Piquer, R. Martin, and P. Company. Using skewed mirror symmetry for optimisation-based 3D line-drawing recognition. *Proc. 5th IAPR Int. Workshop on Graphics Recognition*, pages 182–193, 2003.

[65] L. Ros and F. Thomas. Overcoming superstrictness in line drawing interpretation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(4):456–466, 2002.

[66] J. Rowe, A. Razdan, and A. Simon. Acquisition, representation, query and analysis of spatial data: a demonstration 3D digital library. *Proc. Third ACM/IEEE-CS Joint Conf. on Digital Libraries*, pages 147–158, 2003.

[67] S. Schkolne, M. Pruett, and P. Schroder. Surface drawing: creating organic 3D shapes with the hand and tangible tools. *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, pages 261–268, 2001.

[68] A. Shesh and B. Chen. Smartpaper: An interactive and user friendly sketching system. *Proc. Eurograph*, 2004.

[69] H. Shimodaira. A shape-from-shading method of polyhedral objects using prior information. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(4):612–624, 2006.

[70] I. Shimshoni and J. Ponce. Recovering the shape of polyhedra using line-drawing analysis and complex reflectance models. *Computer Vision and Image Processing*, 65(2):296–310, 1997.

[71] K. Shoji, K. Kato, and F. Toyama. 3-D interpretation of single line drawings based on entropy minimization principle. *IEEE Proc. Computer Vision and Pattern Recognition (CVPR)*, 2:90–95, 2001.

[72] M. Shpitalni and H. Lipson. Identification of faces in a 2D line drawing projection of a wireframe object. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 18(10):1000–1012, 1996.

[73] P. Sturm and S. Maybank. A method for interactive 3d reconstruction of piecewise planar objects from single images. *British Machine Vision Conference*, pages 265–274, 1999.

[74] K. Sugihara. Mathematical structures of line drawings of polyhedrons: toward man-machine communication by means of line drawings. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 4(5):458–469, 1982.

[75] K. Sugihara. An algebraic approach to shape-from-image problem. *Artificial Intelligence*, 23:59–95, 1984.

[76] K. Sugihara. A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(5):578–586, 1984.

[77] K. Sugihara. *Machine Interpretation of Line Drawings.* MIT Press, 1986.

[78] T. Syeda-Mahmood. Indexing of technical line drawing databases. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(8):737–751, 1999.

[79] J.W. Tangelder and R.C. Veltkamp. A survey of content based 3d shape retrieval methods. *Proc. Shape Modeling Applications*, pages 145–156, 2004.

[80] G. Taubin. *A signal processing approach to fair surface design.* Proc. SIGGRAPH, 1995.

[81] A. Turner, D. Chapman, and A. Penn. Sketching space. *Computer and Graphics*, 24:869–879, 2000.

[82] P.A.C. Varley and R.R. Martin. A system for constructing boundary representation solid models from a two-dimensional sketch. *IEEE Proc. Geometric Modeling and Processing*, pages 13–30, 2000.

[83] P.A.C. Varley and R.R. Martin. A system for constructing boundary representation solid models from a two-dimensional sketch—topology of hidden parts. *Proc. First UK-Korea Workshop on Geometric Modeling and Computer Graphics*, pages 113–128, 2000.

[84] P.A.C. Varley and R.R. Martin. Estimating depth from line drawings. *Proc. 7th ACM Symposium on Solid Modeling and Application*, pages 180–191, 2002.

[85] P.A.C. Varley, R.R. Martin, and H. Suzuki. Frontal geometry from sketches of engineering objects: Is line labelling necessary? *Computer Aided Design*, 37:1285–1307, 2005.

[86] P.A.C. Varley, Y. Takahashi, J. Mitani, and H. Suzuki. A two-stage approach for interpreting line drawings of curved objects. *EuroGraphics Workshop on Sketch-Based Input*, pages 105–108, 2004.

[87] A. Vicent, P. Calleja, and R. Martin. Skewed mirror symmetry in the 3D reconstruction of polyhedral models. *Journal of WSCG*, 11(3):504–511, 2003.

[88] D. Vranic. 3d model retrieval. *Ph. D. Dissertation*, 2004.

[89] D. Waltz. *Understanding Line Drawings of Scenes with Shadows*, chapter Psychology of Computer Vision, pages 19–91. New York: McGraw-Hill, 1975.

[90] W. Wang and G. Grinstein. A survey of 3D solid reconstruction from 2D projection line drawings. *Computer Graphics Forum*, 12(2):137–158, 1993.

[91] Y. Wang, Y. Chen, J. Liu, and X. Tang. 3d reconstruction of curved objects from single 2d line drawings. *submitted to the 10th European Conference on Computer Vision (ECCV)*, 2008.

[92] M. Yu, I. Atmosukarto, W. K. Leow, Z. Huang, and R. Xu. 3D model retrieval with morphing-based geometric and topological feature maps. *IEEE. Proc. Computer Vision and Pattern Recognition*, 2:656–661, 2003.

[93] R.C. Zeleznik, K. Herndon, and J. Hughes. Sketch: an interface for sketching 3D scenes. *Proc. SIGGRAPH*, pages 163–170, 1996.