Recovering 3D Geometry from Single Line Drawings

XUE, Tianfan

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

The Chinese University of Hong Kong
July 2011

Abstract

Recovering 3D geometry from a single 2D line drawing is a classic topic in computer vision. The visual system of human beings can interpret a 2D line drawing or a 2D image and perceive its 3D model easily. In order to emulate this ability by a computer vision system, many methods have been proposed in the literature. In this thesis, we study line drawings of manifold objects with hidden lines and vertices visible, as they provide necessary information to reconstruct their complete 3D shapes compared with the line drawings without hidden lines. A line drawing with hidden lines can be obtained from either the sketch of the user on the screen or the scan of the line drawing on a piece of paper. The applications include: virtual reality, user-friendly sketch interface for conceptual 3D designers in CAD systems, 3D query creation for 3D object retrieval, and generating 3D objects from images with user sketches.

In 3D reconstruction from a single line drawing, it normally contains two steps: face identification and 3D reconstruction. Face identification is an important step, since 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. However, most existing algorithms for face identification and 3D reconstruction fail when the line drawing becomes complex. In this thesis, we propose two new approaches that can efficiently solve this problem, especially for complicated line drawings. In face identification part, previous algorithms use an exhaustive searching method, which are too slow to handle a complex line drawing. To solve this problem, we develop an efficient

deduction-based search algorithm for this task, which uses the deduction to cut unnecessary searching branches. To further accelerate our algorithm, we propose several geometric properties and a theorem based on the basic property of manifold objects. Experiments show that our algorithm can deal with more complex objects and is much more efficient than the cutting edge strategies.

In the 3D reconstruction part, we propose a new approach, called object cut, to tackle this problem. Given a complex line drawing representing a manifold object, our algorithm finds the places, called cuts, to separate the line drawing into much simpler ones. The complex 3D object is obtained by first reconstructing the 3D objects from these simpler line drawings and then combining them together. To achieve a natural separation of the object, we propose several propositions and criteria for cut finding. Furthermore, a theorem is given to guarantee the existence and uniqueness of the separation of a line drawing along a cut. The experiments show that the proposed approaches can deal with more complex 3D object reconstruction than state-of-the-art methods. This work has been published on *IEEE Conference on Computer Vision and Pattern Recognition 2010* [31].

摘要

從二維線畫圖中恢復出三維物體是計算機視覺中的一個重要課題。人類視覺系統可以很容易理解二維線畫圖，並且從中推測出它所表示的三維物體。爲了使計算機也能做到這一點，許多研究人員已經提出了各種理解二維線畫圖的方法。在這篇論文中，我們將主要關注與包含不可見部份的線畫圖。相比只包含可見部份的線畫圖，包含不可見部份的線畫圖可以提供更多的信息，而且很容易通過用戶在屏幕或者紙上的繪製得到。這項研究工作的應用背景有：虛擬現實，為CAD設計系統提供一個友好的交互系統，為三維物體檢索提供一個便利的檢索方式，從工程圖或者草圖中恢復三維數據。

從線畫圖中恢復三維信息，一般分成兩個步驟完成：物體錶面檢測以及三維重建。但是之前的物體表面檢測以及三維重建算法，都不能夠處理複雜的線畫圖。物體表面檢測是三維重構中的一個重要步驟，因為如果知道物體由哪些表面構成，重構複雜度將被大大降低。在這篇論文中，我們將提出兩種新的方法，來分別解決在複雜線畫圖中進行表面檢測，以及從中恢復三維信息兩個問題。

關於物體表面檢測這一步驟，之前的算法一般採用暴力搜索的方式尋找表面，所以當二維線畫圖比較複雜的時候，算法難以在一個合理的時間內找到所有的面。爲了解決這個問題，我們提出一個快速的基於推理的搜索算法，使用推理的方法避免了許多不必要的搜索。爲了進一步提高搜索效率，我們提出了一系列關於流型的幾何性質并用這些性質來加速算法。實驗結果表明，我們提出的算法與目前最有效的表面檢測算法進行比較，可以處理更加複雜的二維線畫圖，並且擁有更高的執行效率。

關於三維重構這一步驟，我們提出一種物體割面的算法。對於流型的二維線畫圖，我們首先找到一系列割面，并將線畫圖從這些割面處分開，然後對於每一個分割后的子線畫圖進行三維重構，得到一系列子三維物體。最後將這些子三維物體合並，得到與原線畫圖對應的三維物體。爲了將線畫圖合理的分開，我們提出了一系列性質以及準則來尋找割面。同時我們還證明了對於每一個固定的割面，存在唯一的分割將線畫圖分割開。實驗結果表明，使用物體分割算法比現有算法可以解決更複雜的物體的分割問題。這部份工作已經在 IEEE Conference on Computer Vision and Pattern Recognition 會議上發表[23]。

Acknowledgement

In the past two years, I have an enjoyable and also fruitful life in the Chinese University of Hong Kong and Multimedia laboratory. Many people have land me a hand when I have difficulties in my research career and in my life and their kindness constitute an irreplaceable part of my life in the Chinese University of Hong Kong. I wish to give my most sincere gratitude to these people.

First, I want to give my greatest gratitude to my supervisor Prof. Tang. He is a bright and visionary supervisor. He always comes up some novel ideas and gives me lots of insight suggestions on my research direction. He is also a kind and warm-hearted supervisor. He offers us sufficient room to be an independent researcher and encourages us to find our own way to do research. He always told us that a researcher should have solid background knowledge with an extensive view of state-of-the-art technologies. From Prof. Tang, I not only learn how to deal with a research topic, but I also learn how to be a eligible researcher.

I would also like to thank my advisor, Prof. Liu. He is also the direct supervisor on my research. He is an excellent researcher and keeps a serious attitude towards research, which impresses my a lot. Every time he checks my papers, he always goes through every detail and do not allow any mistakes happens in our work. He is also a very working hard person. Every time I have some problems, he always squeeze some time to solve it, even gives up his sleeping time. His serious and diligence offer me a great help in my research career.

My labmates Wei Zhang, Yueming Wang, Kui Jia, Zhenguo Li, Chen Mo also offered great assistance to my research. They have given me a lot of insight suggestions and teach me how to carry on a research. I benefit from their ways of thinking. I also learn from them some programming skill and fundamental technologies in computer vision and pattern recognition. Without their help, I will spend much more time to find the correct way to carry on my work. Besides, I never forget the memorable life in the multimedia laboratory with all the our labmates: Yueming Wang, Kui Jia, Deli Zhao, Shifeng Chen, Zhenguo Li, Chunjing Xu, Chen Mo, Wei Zhang (both elder and younger ones), Liu Ming, Boqing Gong, Hao Du, Zhimin Cao, Ke Liu, Guangkai Liu, Weiwei Zhang, Meng Wang, Xiaotian He, Rui Zhao, Shi Qiu, Wei Luo, Xixuan Wu, Bolei Zhou. We have a happy and wonderful life in the Chinese University of Hong Kong, playing badminton, basketball, tennis, Shanguosha and swimming together. We also help each others whatever difficulties we are trapped in. Thank you again for your kindly help in these two years.

Last but not the least, I want to thank to my parents and my girlfriend, Li Yanjie. They always support me and their encouragement help me pass many difficulties. Thank you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

2D line drawings interpretation and recovering 3D geometry from 2D line drawings is a classical research topic in computer vision communities. A 2D line drawing is the projection of the wireframe of a 3D object. An example of a line drawing is shown in Figure 1.1(a). A 2D line drawing is the most straight forward way to represent a 3D object, which is widely used in mechanical design and computer-aided design. So it is highly desirable to design a system to interpret a 2D line drawing and recover its 3D geometry. In this paper, we mainly consider a line drawing representing a planar manifold object with hidden lines and vertices visible, as it provides necessary information to reconstruct its complete 3D shape compared with a line drawing without hidden lines. A line drawing with hidden lines can be obtained from either the sketch of the user on the screen or the scan of the line drawing on a piece of paper.

2D line drawing interpretation has many applications. One application of this work is an user-friendly CAD design tool. The user first sketches the wireframe of the 3D object on a screen or on a piece of paper. Then the 3D object is automatically recovered from this wireframe (2D line drawing). The same interface can also be used as for 3D query generation for 3D object retrieval. Another application of this work is to build a 3D object from a single image. Recovering the 3D geometry from a single image is very a difficult task and previous approaches normally need intensive user interactions. Automatic

2D line drawing interpretation algorithm provides an easy way to recover 3D geometry from a single image. The user first sketches the wireframe of the object on the image and the 3D object is then generated from this wireframe (2D line drawing). The final 3D object is obtained by mapping the texture on the image to the 3D object. Correctly recovering 3D geometry from the line drawing is the key to these approaches and greatly affects the quality of reconstructed 3D object [1], [6], [11], [13], [20] , [28], [10], [24], [12].

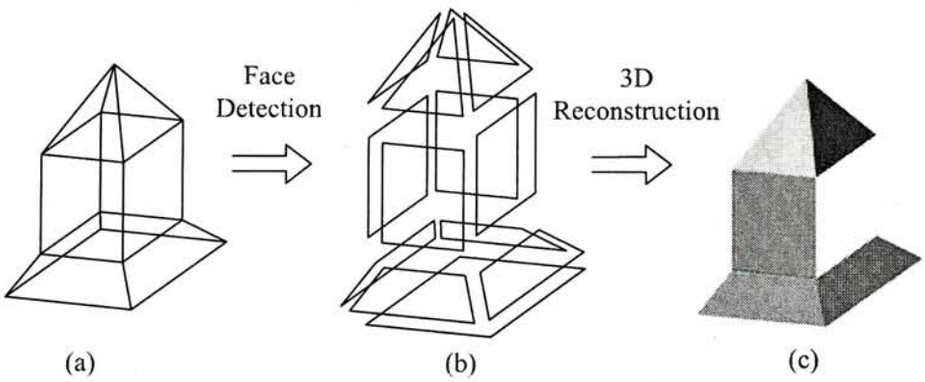


Figure 1.1: Pipeline of recovering 3D geometry from a line drawing. (a) An input line drawing. (b) The face configuration of (a). (c) Reconstructed 3D object.

Although the visual system of the human beings can easily understand a 2D line drawing, 2D line drawing interpretation and recovering 3D geometry from 2D line drawings is a challenge task for computer. In past decades, many works have been proposed to interpret a 2D line drawing and reconstruction the 3D model from the line drawing [6], [13], [20], [11], [16], [25], [30], [21], [22], [26], [27], [29]. In these works, the 3D reconstruction from a single line drawing is normally decomposed into two steps: face identification and 3D reconstruction. Using this method, a correct 3D object can easily be recovered from a simple line drawing with few lines and vertices. However, when the line drawing becomes complex, most existing algorithms fail, either because the computation cost is too high to get a result in a reasonable time, or because it is easily to be trapped in the local minimum when finding the optimal solution.

1.1 Previous Approaches on Face Identification

Previous works on reconstruction from 2D line drawing consists of two steps: face identification and 3D reconstruction [6], [20], [11], [16], [25], [13], [30]. A 3D object consists of faces. If the face configuration of an object is known before the reconstruction, the complexity of the reconstruction will be reduced significantly. So most of previous works first find the face configuration, and then recover the 3D geometry from a 2D line drawing based on the detected face configuration. Figure 1.1(b) shows the face configuration of the line drawing shown in Figure 1.1(a), and the recovered 3D object is shown in Figure 1.1(c). The details of each step will be introduce as follows:

1.1.1 Face Identification

There are two kinds of 3D objects: manifold 3D object and non-manifold 3D object. A manifold, or more specifically 2-dimensional manifold[1] , is a solid where every point on its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space [2]. A manifold is an oriented 3D object, which means it separates the whole 3-dimensional Euclidean space into two regions: the region inside the object and the region outside the object, and its volume can be measured by the volume of the inner regions. The object which is not a manifold is non-manifold object. Non-manifold object normally consists a set of pieces of sheets.

Figure 1.2(a) shows two examples of manifold object and Figure 1.2(b) shows an example of non-manifold object. The first row shows the examples of input line drawings and the second row shows its corresponding 3D models. Notice that the non-manifold object in Figure 1.2(b) is not a solid object and

[1]For the ease of description, we simply refer to the 2-dimensional manifold as *manifold.*

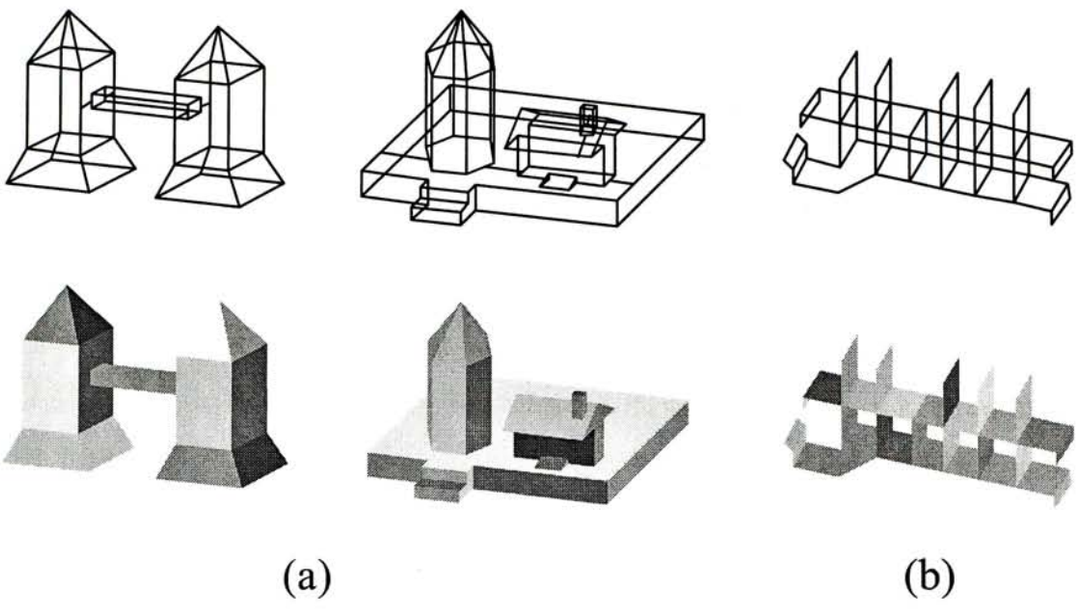


Figure 1.2: Example of line drawings representing manifold (a) and non-manifold (b) objects.

only consists of a set of sheets.

Recent face identification methods are either for general objects [16], [25] or for manifold objects [15], [9]. A general object may either be manifold or non-manifold, so it is considered to be formed by a set of sheet patches (faces) without considering if it is a solid or not. The algorithms for sheet objects in [16] and [25] try to find as many faces as possible even if they are invisible internal faces of a solid. Here an invisible internal face denotes those faces that are totally inside the object. For example, in Figure 1.4(c), the two faces marked by bold lines are internal faces, since they are invisible from the outside of the object.

1.1.2 General Objects

The face detection algorithms for general objects normally form the face detection problem as an optimization problem. In [25], the authors first find a set of potential faces from a given line drawing using the circuit space method.

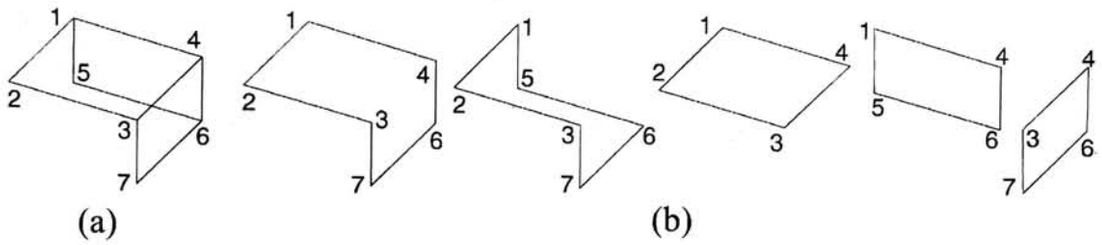


Figure 1.3: The potential faces set and real face set of a line drawing. (a) An input line drawing. (b) Five potential faces found by the algorithm. Only the last three faces are the real faces.

For example, the potential faces of the line drawing shown in Figure 1.3(a) found by [25] is shown in Figure 1.3(b). Then the real face set X' are selected from the potential face set X by maximizing a target function $g(x)$, subjecting to that all the faces in X' can coexist. Formally, the real face selection problem from a potential face set X can be formed as the following optimization problem [25]:

$$\begin{aligned} \min \quad & g(X') = \sum[R^+(e) - R(e)] + \sum[R^+(v) - R(v)] \qquad (1.1) \\ \text{subject to:} \quad & R(e) \leq R^+(e), \forall e, \\ & R(v) \leq R^+(e), \forall v, \\ & \text{faces in } X' \text{ can coexist.} \end{aligned}$$

where $R(e)$ and $R(v)$ are the actual edge and vertex ranks in X' respectively, $R^+(e)$ and $R^+(v)$ are the upper bound of edge and vertex ranks of the line drawing. The detail of derivation of $R^+(e)$ and $R^+(v)$ are discussed in [25]. The authors model the face identification problem as to find a subset X' of the potential face set X as to maximize the edge rank and vertex rank of the drawing, as long as the faces in X' can coexist. Here two faces cannot coexist if they share two edges which are not collinear. The optimization problem (1.1) is in general a NP-hard problem, so the authors in [25] propose a $A*$ algorithm to find its optimal solution. For a simple line drawing with few vertices, this algorithm can find its face configuration. However, when the line drawing

becomes complex, this algorithm normally takes unacceptable long time to find all its faces.

To accelerate the algorithm, in [14], the authors prove that the equation (1.1) can be further simplified as follows:

$$\begin{aligned} \max \quad & f(X') = \sum_{i \in X'} d(i), \text{ where } X' \subset X \\ \text{subject to:} \quad & \text{faces in } X' \text{ can coexist.} \end{aligned} \tag{1.2}$$

where $d(i)$ is the number of vertices in face i. The authors model the face identification problem as to find a subset X' of the potential face set X as large as possible, as long as the faces in X' can coexist. Comparing with (1.1), (1.2) has less constraints and can be solved more efficiently. The authors propose to remodel the optimization problem (1.2) as a maximum clique problem and solve it using the branch and bound strategy. This algorithm is much faster than the one in [25]. However, it is still an algorithm with exponential running time and cannot deal with complicated line drawings.

To further accelerate the algorithm, in [16], the authors proposes to solve the problem in (1.2) using a variable-length genetic algorithm. A heuristic rule and geometric constraint is proposed to do a local search. Since evolutionary search is considered to be one of efficient way to find the optimal solution for a NP-hard problem, the proposed algorithm is much faster than the previous methods. The experiment also shows that this algorithm suffers little of the exponential explosion when the line drawing grows complex.

The face identification algorithm for general object can also be used to find faces of manifold object. However, when we want to identify the faces of a manifold, we only want to have the faces on its surface. For example, the faces in Figure 1.4(c) are obtained by these algorithms where the two internal faces (bold) are undesirable. Furthermore, the face detection algorithms for general objects cannot detect hole cycles in a line drawing of a manifold; instead, they

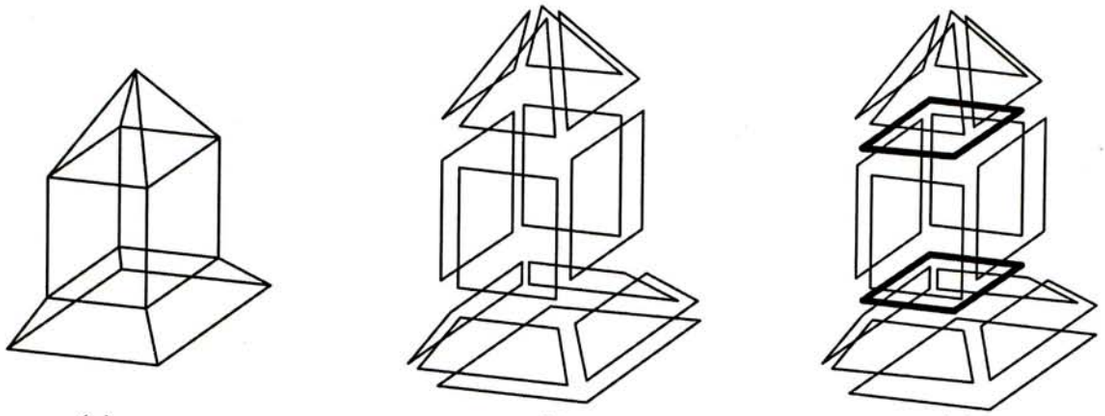


Figure 1.4: Differences between the face identification method for manifold objects and general objects.

output them as faces.

1.1.3 Manifold Objects

The algorithms in [15] and [9] are for face identification from line drawings of manifolds. Liu et al.'s algorithm [15] is a global search algorithm consisting of two steps: searching for cycles from a line drawing and searching for real faces from these cycles. The first step uses a depth first search strategy to find all the cycles from a line drawing. The number of cycles found by the depth first search increases exponentially with the increase of the number vertices in the line drawing. And in the second step, the key to find a real face set is that each edge of the line drawing is passed by exactly two real faces. Based on this geometry property, Liu et al. use an exhaustive tree search method to find a set of faces which passes each edge exactly twice. This step also suffers the exponential explosion problem. Therefore, when dealing with a complex manifold, the combinatorial explosion renders this method difficult to obtain the result in an acceptable time, and thus prevents it from practical use.

Li et al.'s algorithm [9] is a local search algorithm, which first finds out all the faces formed from a local set of vertices and then gradually enlarges the

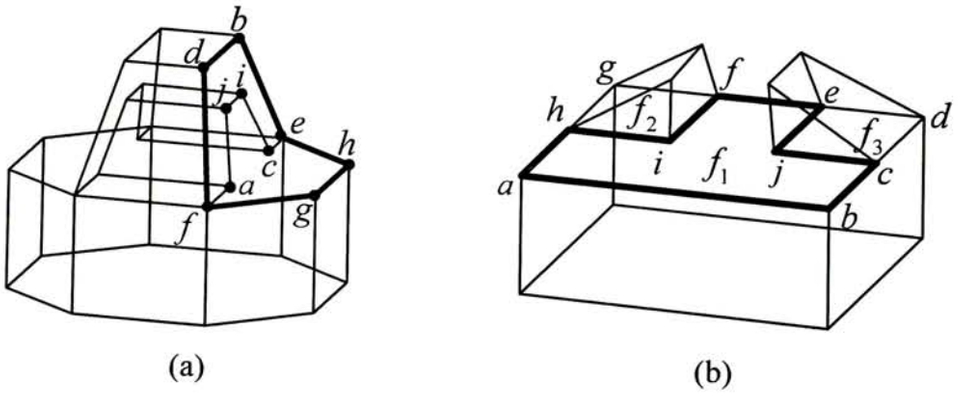


Figure 1.5: A fail example of the algorithm in [9]. (a) A line drawing from which the algorithm in [9] fails to find the real faces, where the bold cycle is mistakenly considered as a face. (b) Another line drawing which this algorithm cannot handle, where the bold cycle is not detected.

set to find more faces. The authors propose a greedy search rule where a cycle with fewer edges has a higher priority to become a face. However, this rule may make the search trapped in local optima. Take the line drawing shown in Figure 1.5(a) as an example. In a local vertex set $\{d, b, j, i, f, a, c, e, g, h\}$, cycle (f, g, h, e, b, d, f) is mistakenly considered as a face by the algorithm since it has the fewest edges among all the cycles in this vertex set. This error further prevents the detection of the real face $(a, j, i, c, e, b, d, f, a)$ and results in wrong face identification. Figure 1.5(b) shows another example where this algorithm cannot obtain the correct result. It fails to find cycle $f_1 = (a, h, i, f, e, j, c, b, a)$ when cycles $f_2 = (h, g, f, i, h)$ and $f_3 = (c, d, e, j, c)$ are found, because f_1 is blocked by f_2 and f_3. A cycle is not considered as a face if it is blocked by more than two faces [9].

1.2 Previous Approaches on 3D Reconstruction

Previous works on the line drawing interpretation consider a line drawing as an orthogonal projection of a 3D wireframe of an object. Suppose there is a 2D line drawing L and we need to recover the corresponding 3D object O. The 3D coordinate of the vertices of a 3D object O are denoted as $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$, and 2D orthogonal projections of these 3D coordinates are denoted as $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ which can be got from the input line drawing (n is the number of vertices in the object). Then recovering 3D geometry of a line drawing equals to estimating the z coordinates $(z_1, z_2, \ldots, z_n)$ from the input x and y coordinates $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.

Recovering the 3D geometry from a 2D line drawing is an ill-pose problem. Given a 2D line drawing, there are many corresponding 3D objects whose 2D projections equal to this line drawing. For example, both the 3D objects shown in Figure 1.6(b) and Figure 1.6(c) have the same projections as the line drawing shown in Figure 1.6(a) [11]. However, only one interpretation is most consistent with human's interpretation. For example, given a line drawing shown in Figure 1.6(a), the 3D object shown in Figure 1.6(c) is more reasonable and more consistent with human's interpretation than the 3D object shown in Figure 1.6(b). Therefore recovering the 3D geometry from a 2D line drawing is to find the 3D object that is most consistent with human's understanding from all these possible 3D objects.

Given a 2D line drawing, each 3D object whose 2D projection equal to this line drawing can be uniquely defined by a n-dimensional vector $(z_1, z_2, \ldots, z_n)$ representing its z coordinate. Therefore, recovering the 3D geometry from a line drawing is modeled as the following optimization problem [19].

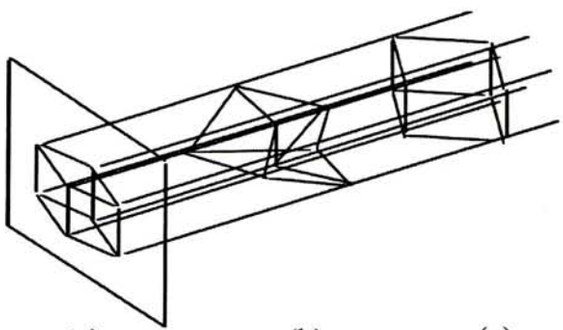


Figure 1.6: Ambiguity in recovering 3D geometry from a line drawing. (a) An input line drawing. (b) One interpretation of this line drawing. (c) The other interpretation of this line drawing.

$$\min_{z_1,z_2,\ldots,z_n} f(z_1, z_2, \ldots, z_n) \tag{1.3}$$

where $f(z_1, z_2, \ldots, z_n)$ evaluates how a 3D object represented by the vector $(z_1, z_2, \ldots, z_n)$ is consistent with human's interpretation.

Many works have been proposed [8], [11], [19] to discuss how to define the optimization target function $f(\cdot)$ to better evaluate a 3D object. In [19], Marill proposes a minimum standard deviation of angles (MSDA) principle, that is a good 3D reconstruction should minimize the standard deviation of the angles in the reconstructed object. In [8], Leclerc and Fischler claim that for many 2D line drawing, MSDA is not enough to correctly recover its 3D geometry. So they propose first to detect the planar faces in the line drawing and then to enforce that the vertices on the same face should lying on the same plane, which is called the face planarity principle. Notice that the face configuration of the line drawing required in this step is already detected using the method discussed in section 1.1. In [11], Lipson and Shpitalni propose 13 image regularities to define the optimization function in (1.3), including face planarity, line parallelism, line verticality, isometry, corner orthogonality, skewed facial orthogonality, skewed facial symmetry, line orthogonality, MSDA, face perpendicularity, prismatic face, line collinearity, planarity of skewed chains. Using these principles and

image regularities to define an optimization function, this method can correctly recover the 3D geometry of a simple line drawing.

However, when a line drawing becomes complex, most previous algorithms [3], [4], [5], [8], [11], [13], [19] fail in the reconstruction, getting trapped in local minima due to the large number of variables in the objective functions [17]. To solve this problem, some researchers propose to separate a complex line drawing into multiple simpler line drawings, then independently reconstruct the 3D objects from these line drawings, and finally merge them to form a complete object [4], [17]. This approach well solves the problem mentioned above and using this decomposition approach, the one in [17] can handle most complex objects comparing with the previous method. Its key step is how to separate a complex line drawing. The authors propose to do it from the *internal faces* of the line drawing. An internal face is a face inside an object with only its edges visible on the surface and these edges are all from the line drawing. However, this method may fail when a complex object has no or too few internal faces. One example is given in Figure 1.7(a) where there is only one internal face (Figure 1.7(b)), from which the separation is shown in Figure 1.7(c). We can see that the bigger object in Figure 1.7(c) is still complex.

1.3 Our approach for Face Identification

In this thesis, we propose a novel method to efficiently find the faces from the line drawing representing a manifold object. Our algorithm is based on the result obtained by the algorithm in [16]. Given a line drawing of a manifold, the algorithm in [16] outputs a set of faces including some internal faces and all or most of the real faces. We do not find all the cycles in the line drawing, which is time-consuming and normally there are tremendous cycles which make the real face selection hard to process. Instead we use the output of [16] as the

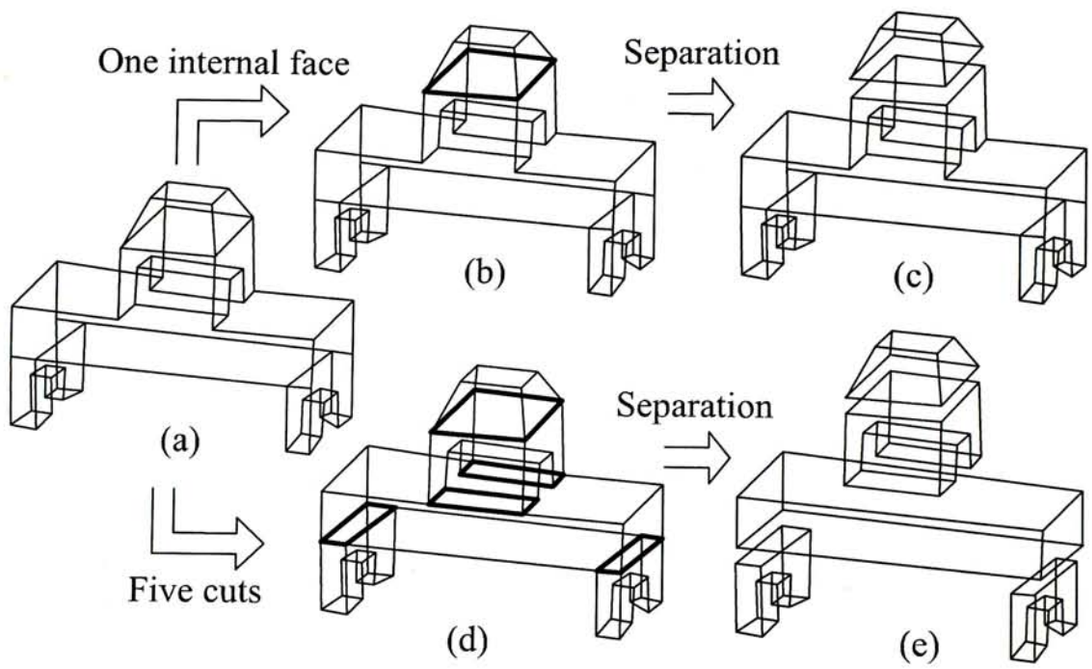


Figure 1.7: Comparison between the method in [17] and object cut. (a) A line drawing. (b) Only one internal face. (c) Separation result by [17]. (d) Five cuts. (e) Separation result by object cut.

set of potential faces, which is less time consuming and less potential faces are found. To find the real face set from these potential faces, we develop a fast deduction-based search method to find all the real faces with the internal faces removed. Unlike previous approaches [14], [25] using a brutal-force search method to find the correct face configuration, which suffers an exponential exploration when the line drawing become complex, the execution time of our algorithm increase nearly linearly with the increase of the number of vertices. To further remove the errors in the faces found by [16], we propose several geometric properties and a theorem. Our experimental results show that our algorithm can deal complex line drawings of manifolds much more efficiently than previous algorithms.

1.4 Our approach for 3D Reconstruction

In this thesis, we propose a novel approach, called *object cut*, to decompose a complex line drawing into multiple simpler line drawings. We use cuts but not internal faces to partition a line drawing. An internal face is a special case of a cut. One example is shown in Figure 1.7. From Figure 1.7(a), our algorithm can find five cuts (Figure 1.7(d)), based on which the line drawing is separated into five simpler ones (Figure 1.7(e)). Note that only one of these cuts is an internal face since the other four cuts contain edges not from the original line drawing. Obviously, the reconstruction problem from the line drawings in Figure 1.7(e) is easier to handle than those in Figure 1.7(c). We develop several propositions and criteria for cut finding, and present a theorem showing the existence and uniqueness of the separation of a line drawing along a given cut. Our experimental results indicate that our approach can deal with 3D reconstruction of more complex objects than previous methods. This work has been published on *IEEE Conference on Computer Vision and Pattern Recognition 2010* [31].

Chapter 2

Face Detection

As mentioned in Chapter 1, recovering 3D geometry from a line drawing consists of two steps: face detection and 3D reconstruction. In this chapter, we focus on face detection step. we develop an efficient deduction-based search algorithm for this task. Unlike previous methods [14] [25] using a brutal-force search to find the correct face configuration, which suffers exponential exploration of execution time when line drawing becomes complex, we propose a novel method to efficiently find the real faces from the line drawing of a manifold, by exploiting the result obtained by the algorithm in [16]. We use it but not the one in [25] because the former is much more efficient than the latter while they generate the same result. In what follows, GAFI stands for the algorithm in [16] since it uses a genetic algorithm for face identification. Given a line drawing of a manifold, GAFI outputs a set of faces including some internal faces and all or most of the real faces. Based on this set and the line drawing, we develop a fast deduction-based search method to find all the real faces with the internal faces removed. Several geometric properties and a weak face adjacency theorem are presented to help the search. Our experimental results show that our algorithm can deal with more complex objects and is much more efficient than the state-of-the-art method [15] for face identification from line drawings of manifolds.

2.1 GAFI and its Face Identification Results

Because GAFI [16] is used to find the initial set of faces for our algorithm, we give more description of it in this section. Although GAFI is designed for face identification from line drawings of sheet objects, due to its efficiency, we still use it to detect faces from line drawings of manifolds by treating the manifolds as sheet objects. Before proceeding to the description of GAFI, we first list several terms that are extensively used in the rest of the paper.

- **Manifold**. A manifold, or more specifically 2-dimensional manifold, is a solid where every point on its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space [2]. Manifolds considered in this paper are made up of flat faces. One property of planar manifold is that each edge is shared exactly by two real faces [7].
- **Real face**. A real face is a face on the surface of the manifold.
- **Internal face**. An internal face is a face inside a manifold with only its edges visible on the surface [4].
- **Face**. A face is either a real face or an internal face.
- **Difference of two faces**. In a 2D line drawing, if face f_1 is enclosed by another face f_2, the difference of them is a cycle whose boundary is formed by f_1 and f_2 with their common edges removed.
- **Potential face cycle (PFC)**. A PFC is a non-self-intersecting cycle without any edge connecting two of its non-adjacent vertices.

In Figure 2.1(b), all the cycles are real faces, while in Figure 2.1(c), the two bold cycles are internal faces. In Figure 2.2(a), the difference of two faces (a, b, i, h, j, k, a) and (k, l, m, j, k) is a cycle $(a, b, i, h, j, m, l, k, a)$. In [16], a PFC is called a minimal potential face. In order not to confuse it with a face, we use a different term in this paper. As in [4], we also consider an internal face

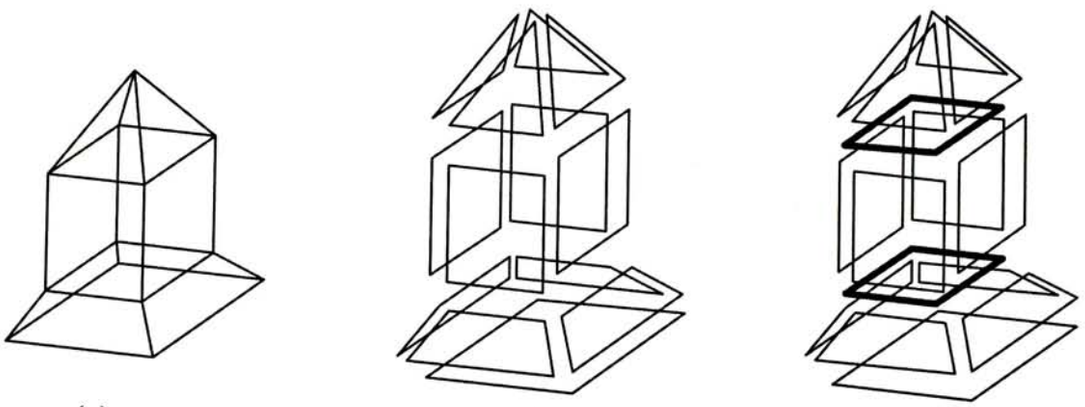


Figure 2.1: Differences between the face identification method for manifold objects and general objects.

as planar, because it is formed by gluing two planar manifolds or subtracting a planar manifold from another.

In GAFI, face identification is formulated as an optimization problem: finding a face set as large as possible, as long as the found faces can coexist in the same object. Each face is a PFC. The coexistence of two faces is constrained by a face adjacency theorem, which states that two adjacent planar faces can coexist in the same object if and only if their common vertices are colinear [16]. For example, in Figure 2.2, cycles (m, n, o, k, l, m) and (o, k, l, o) cannot both be faces since they share three non-colinear vertices. This theorem is obvious because the intersection of two planar faces is on a straight line. In this paper, this theorem is called the *strong face adjacency theorem* in order to distinguish it from the *weak face adjacency theorem* defined in Section 2.2.2.

GAFI is able to efficiently find all the faces from the line drawing of a sheet object whose faces are all considered as real faces. Therefore, when it is applied to the line drawing of a manifold, the found face set usually contains both real faces and undesirable internal faces. Besides, GAFI may miss some real faces of the manifold. For example, two real faces in bold cycles in Figure 2.2(a) do not appear in its result (Figure 2.2(b)). Through our experiments, we find

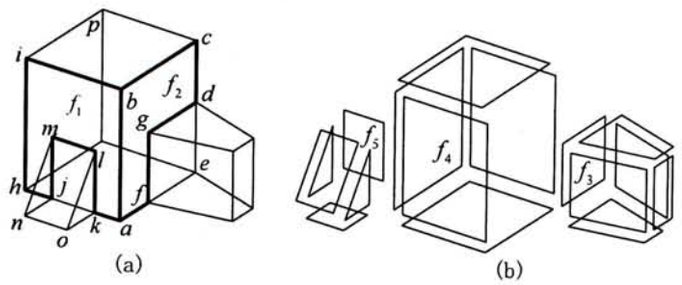


Figure 2.2: (a) A line drawing. (b) Faces found by GAFI. GAFI fails to find the real faces (a, b, c, d, g, f, a) and $(h, i, b, a, k, l, m, j, h)$ if (a) represents a manifold.

that there are two cases where GAFI may miss a real face. One is that when an internal face shares some non-colinear vertices with a real face, the real face may be missed, which is called *type 1 error*. For example, in Figure 2.2, the real face $f_2 = (a, b, c, d, g, f, a)$ cannot be detected if the internal face $f_3 = (d, e, f, g, d)$ is found, due to the strong face adjacency theorem. From this example, we can see that this theorem is too restrict to include all real faces in its result for a manifold.

Another case is that when a real face of a manifold is the difference of another two faces, the real face is replaced by these two faces in the result, which is called *type 2 error*. For example, in Figure 2.2, since the difference of faces $f_4 = (h, i, b, a, k, j, h)$ and $f_5 = (m, l, k, j, m)$ is $f_1 = (h, i, b, a, k, l, m, j, h)$, the real face f_1 is missing while f_4 and f_5 are in the face set found by GAFI (see Figure 2.2(b)). The appearance of this error is because GAFI tries to maximize the number of found faces.

2.2 Our Face Identification Approach

In this section, we propose a new approach to face identification from line drawings of manifolds based on the results obtained by GAFI. Since GAFI treats a line drawing as a sheet object, the faces found by GAFI include real

faces and internal faces. Besides, GAFI may miss some real faces in its result, as shown in Figure 2.2. In Section 2.2.1, we discuss how to distinguish real faces from internal faces. In Section 2.2.2, we introduce the weak face adjacency theorem, which provides a looser constraint on face coexistence. In Section 2.2.3 and Section 2.2.4, we present solutions to the problems caused by the type 1 and type 2 errors, respectively.

2.2.1 Real Face Detection

With the face set F_{found} found by GAFI, we develop a deduction-based search method to find the real face set F_{real} from F_{found}. In this section, we assume that there are no missing real faces in F_{found}, which is called the *real face completeness assumption.* Since the schemes proposed in Sections 2.2.3 and 2.2.4 are able to find the missing real faces, we can always assume that F_{found} contains all the real faces. Next, we introduce two properties first.

Property 1. *Each edge of a line drawing is shared exactly by two real faces.*

Property 2. *Given a line drawing and the face set F_{found} found by GAFI, (a) if an edge is shared exactly by two faces in F_{found}, then these two faces are real faces; (b) if an edge is already passed by two real faces, then all other faces passing through this edge are internal faces.*

Property 1 is a property of manifolds [7]. Property 2 is a direct corollary of Property 1 and the real face completeness assumption. Property 2 is used to determine whether a face is a real face or an internal face. For example, given a face set found by GAFI as shown in Figure 2.3(b), according to Property 2(a), faces (a, i, b, a) and (a, i, d, a) are real faces since they are the only two faces passing through edge (a, i). Faces (a, e, h, d, a) and (a, b, f, e, a) are also real faces since edge (a, e) is passed only by these two faces. Face (a, b, c, d, a) is an internal face according to the Property 2(b), since edge (a, b) is already passed

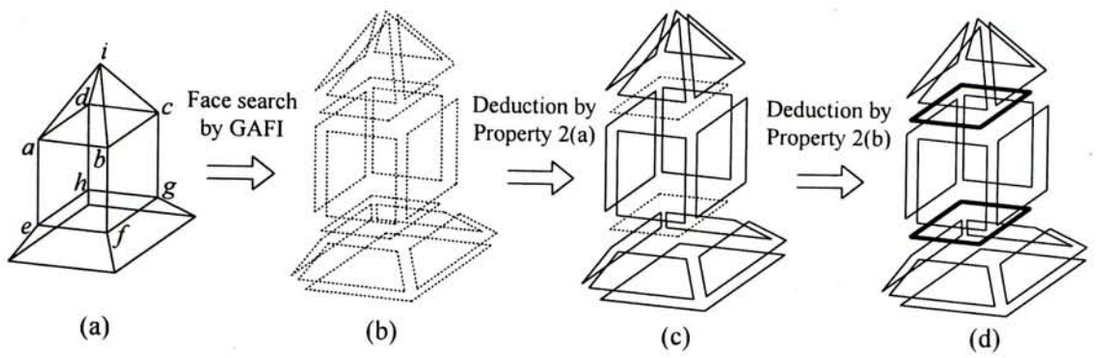


Figure 2.3: An example of real face detection where the real faces are marked by thin cycles, the internal faces are marked by bold cycles, and unknown faces are marked by dashed cycles. (a) Input line drawing. (b) Faces found by GAFI. (c) Result after the deduction using Property 2(a). (d) Result after the deduction using Property 2(b).

by two real faces (a, i, b, a) and (a, b, f, e, a). Based on Property 1, the real face detection problem is formulated as follows:

Definition 2.1. *Given a line drawing and a face set F satisfying the real face completeness assumption, the real face detection problem is formulated as finding a subset F' of F such that every edge of the line drawing is passed by exactly two faces in F'.*

We develop an efficient deduction-based search algorithm to solve this problem, which is outlined in Algorithm 1. At the beginning of the algorithm, all the faces in F_{found} belong to the 'unknown' set $F_{unknown}$, meaning that all these faces are unknown to be real faces or internal faces initially. Then steps 1–7 try to deduce whether a face is a real or internal face using Property 2. Steps 3 and 4 are based on Property 2(a), and steps 5 and 6 are based on Property 2(b). The algorithm is fast; after several iterations, each face is classified as either a real face or an internal face. Figs. 2.3(c) and (d) illustrate the deduction on the line drawing in Figure 2.3(a).

Algorithm 1 Real face detection

Input: A line drawing $G = (V, E)$ and the face set F_{found} found by GAFI
Call $FindRealFaces(G, \phi, F_{found})$
Output: The real face set found by the following procedure

Procedure: $FindRealFace(G, F_{real}, F_{unknown})$

1. **do**
2. **for** every edge $e \in E$
3. **if** e is shared by exactly two faces in $F_{unknown} \cup F_{real}$
4. Move all the faces passing through e from $F_{unknown}$ to F_{real}
5. **else if** e is shared by two faces in F_{real}
6. Delete all the faces passing through e from $F_{unknown}$
7. **while** $F_{unknown} \neq \phi$
8. **return** the real face set F_{real}

2.2.2 The Weak Face Adjacency Theorem

As mentioned in Section 2.1, it is the strong face adjacency theorem that causes the type 1 error. In this section, we propose a weak face adjacency theorem to solve this problem. We first define a new term before giving this theorem.

Definition 2.2. *A face set $\mathcal{F}$ of a line drawing is called a coplanar face set if the faces in $\mathcal{F}$ can be written as $f_1, f_2, \ldots, f_m$ such that: for $j = 2, \ldots, m$, the vertex set $Vertex(f_j) \cap Vertex(\{f_t\}_{t=1}^{j-1})$ contains at least three non-colinear vertices, where $Vertex(f_j)$ contains all the vertices of face f_j and $Vertex(\{f_t\}_{t=1}^{j-1})$ contains all the vertices appearing in the face set $\{f_t\}_{t=1}^{j-1}$.*

Property 3. *All faces in the same coplanar face set lie on the same plane in 3D space.*

Based on Definition 2.2, this property is obvious because in a coplanar face set, each face shares at least three non-colinear vertices with at least one other face. In Figure 2.4(a), three faces $f_1 = (a, b, m, l, j, k, d, a)$, $f_2 = (l, m, i, j, l)$, and $f_3 = (i, c, k, j, i)$ form a coplanar face set.

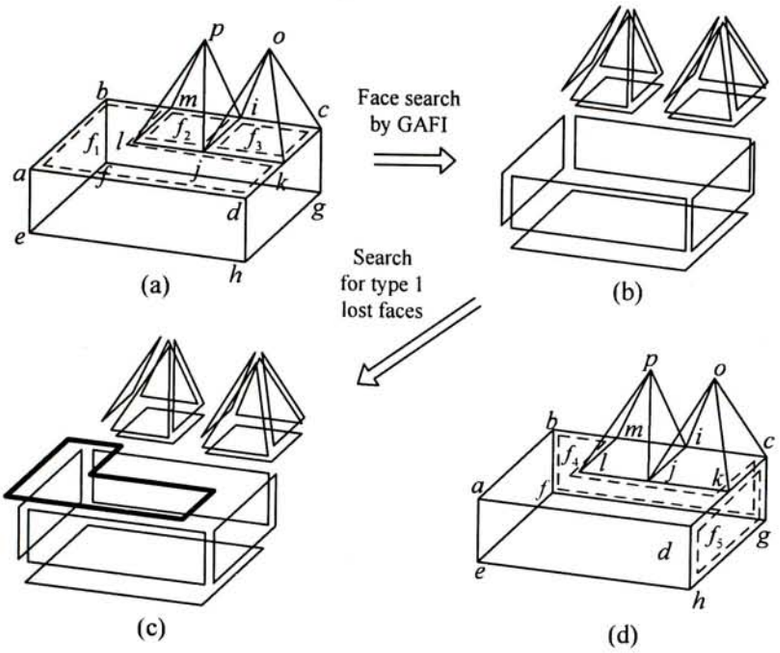


Figure 2.4: (a) A line drawing where a coplanar face set $\{f_1, f_2, f_3\}$ is denoted by dashed cycles. (b) The face set found by GAFI. (c) The face set obtained by Algorithm 2, where the lost real face (bold) is found. (d) Two cycles f_4 and f_5 (dashed) that cannot coexist according to the weak face adjacency theorem.

Theorem 2.3. (Weak face adjacency theorem) *Let F be a face set of a line drawing. Two faces $f_i, f_j \in F$ cannot coexist if they belong to the same coplanar face set and have an overlapping region in the 2D line drawing plane.*

Proof. According to Property 3, f_i and f_j lie on the same 3D plane. Furthermore, since they have an overlapping region in the 2D line drawing plane, they must have an overlapping region in 3D space. Therefore, they cannot coexist in the same object because the faces of an object do not overlap with each other in 3D space. □

The weak face adjacency theorem provides a looser constraint on a face set than the strong face adjacency theorem. For example, for the line drawing shown in Figure 2.4(a), the strong face adjacency theorem does not allow f_1 to coexist with f_2 or f_3, but the weak face adjacency theorem does. Therefore, with the weak face adjacency theorem, we can add faces, which are lost due to

the type 1 error, to the face set found by GAFI. How to search for these lost faces is discussed in the next section.

2.2.3 Searching for Type 1 Lost Faces

Based on the weak face adjacency theorem, we propose a depth-first search algorithm (Algorithm 2) to find lost faces due to the type 1 error, which are called type 1 lost faces. This algorithm tries to add as many PFCs as possible to the set of faces found by GAFI, as long as the PFCs can coexist with these faces according to the weak face adjacency theorem. In Algorithm 2, the search for type 1 lost faces is carried out by examining the edges of the line drawing one by one through a procedure $ExtDFS$. $ExtDFS$ is an extended version of a common depth-first search (DFS) algorithm in [18]. The difference between them is that the DFS algorithm is used to find all cycles in a graph (line drawing here), while $ExtDFS$ finds all those cycles that are PFCs and can coexist with all the faces found by GAFI according to the weak face adjacency theorem. $ExtDFS$ is not difficult to design based on the DFS algorithm by incorporating the constraints imposed by a PFC and the weak face adjacency theorem. It is omitted here due to the space limitation.

Algorithm 2 Type 1 lost face search

Input: A Line Drawing $G = (V, E)$ and the face set F_{found} found by GAFI

1. **for** each edge $e_i = (v_{i,1}, v_{i,2}) \in E$
2. Set the current path $p \leftarrow e_i$ and delete edge e_i from E
3. Call $ExtDFS(v_{i,1}, v_{i,2}, p, F_{found})$

Output: The face set F_{found} with type 1 lost faces added

Figure 2.4(c) shows the result after performing Algorithm 2 with the initial face set F_{found} as shown in Figure 2.4(b). The bold cycle in Figure 2.4(c) is the only type 1 lost face found by $ExtDFS$.

The weak face adjacency theorem can be used to solve the type 1 error

problem for two reasons: First, those real faces violating the strong face adjacency theorem can still be detected using the weak face adjacency theorem. For example, face f_1 in Figure 2.4(a) is found and added to the face set (Figure 2.4(c)) although it shares three non-colinear vertices with face f_2. Second, it can efficiently eliminate those PFCs that cannot coexist with the faces found by GAFI. For example, in Figure 2.4(d), the PFC $f_4 = (b, m, l, j, k, c, g, f, b)$ is not added to the face set in Figure 2.4(c) because it cannot coexist with face $f_5 = (c, g, h, d, k, c)$ even under the constraint of the weak face adjacency theorem.

2.2.4 Searching for Type 2 Lost Faces

With the line drawing in Figure 2.5(a), let us recall the type 2 error: In the face set found by GAFI (Figure 2.5(b)), a real face $(a, b, j, i, c, d, e, f, a)$ is replaced by two other faces (a, b, c, d, e, f, a) and (b, c, i, j, b) whose difference is the real face, which is called a type 2 lost face. In this section, we propose a scheme to find these lost faces. For simplicity of the following description, when we say the difference of a face pair, we mean the difference of the two faces of the pair (see Section 2.1 for the definition of the difference of two faces).

We first check every face pair in the face set F found by GAFI together with type 1 lost faces added in order to find those pairs whose differences have the potential to be real faces. Then we add the differences of these face pairs to F. The following property is used to check whether the difference of a face pair has the potential to be a real face.

Property 4. *Given a line drawing and the face set F found by GAFI with type 1 lost faces added, if the difference of a face pair $\{f_1, f_2\} \subset F$ is a real face of the line drawing, then (a) the difference of f_1 and f_2 is a PFC, and (b) each of the common edges of f_1 and f_2 is passed by at least two other faces in F besides f_1 and f_2.*

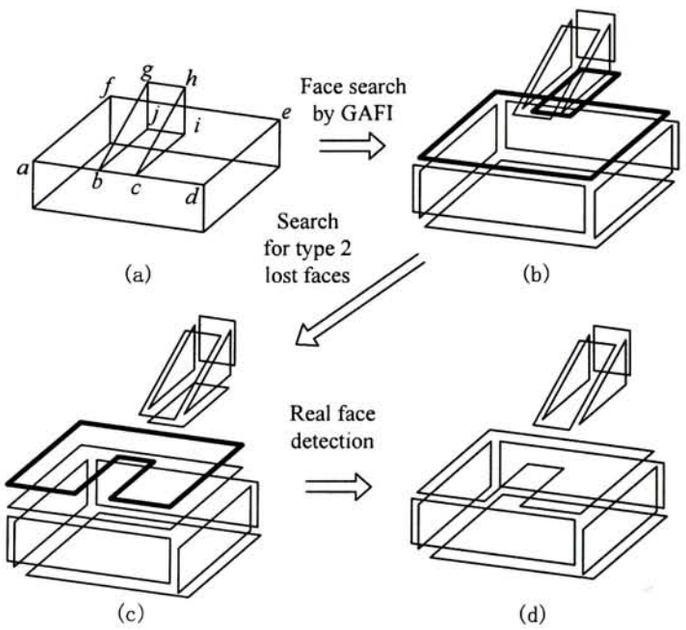


Figure 2.5: (a) A line drawing. (b) The face set found by GAFI. (c) The face set after searching for type 2 lost faces. (d) The face set after the real face detection.

If the difference of two faces is not a PFC, it cannot be a face, which verifies Property 4(a). If a common edge e of f_1 and f_2 is passed by at most one other face in F besides f_1 and f_2, we cannot find two real faces from F that pass through e since f_1 and f_2 cannot be real faces when their difference is a real face. In this case, the edge e is not passed through by two real faces, which contradicts Property 1, and thus Property 4(b) is verified.

In order not to miss a type 2 lost face, we check every face pair $\{f_i, f_j\}$ from F. If f_i is enclosed by f_j (or f_j is enclosed by f_i) and the face pair $\{f_i, f_j\}$ satisfies the two conditions in Property 4, the difference of f_i and f_j is added to F. Since Property 4 is a necessary condition, all the face pairs whose differences are real faces (type 2 lost faces) cannot be missed in this process. Although some PFCs that are not real faces may be included in F, they are eliminated by the real face detection in Algorithm 1. Note that when adding the difference of a face pair to F, the two faces of the pair are still kept in F in case they are real faces. If they are not real faces, they can also be eliminated

Algorithm 3 Complete face identification algorithm

1. Use GAFI to find an initial face set F_{found}
2. Search for type 1 lost faces and add the found PFCs to F_{found}
3. Search for type 2 lost faces and add the found PFCs to F_{found}
4. Perform real face detection from F_{found} and return the real faces

in Algorithm 1.

Figure 2.5 illustrates the procedure of searching for type 2 lost faces. The face pair (marked in bold in Figure 2.5(b)), whose difference has the potential to be a real face according to Property 4, is found, and the difference is added to F (Figure 2.5(c)). The two faces of this face pair are deleted after the real face detection (Figure 2.5(d)). The outline of our complete face identification algorithm is given in Algorithm 3.

2.3 Experimental Results

We have conducted extensive experiments to verify the performance of our algorithm. As mentioned previously, there are four recent face identification algorithms [16], [25], [15], [9]. Since the algorithms in [16] and [25] are for sheet objects (but not suitable for manifolds) and the algorithm in [9] cannot handle many common line drawings of manifolds such as the two shown in Figure 1.5, we compare our algorithm with the one in [15] only. In what follows, SSTS stands for the algorithm in [15] since it uses a state-space tree to search for real faces, and DBS stands for our algorithm that performs deduction-based search. Both algorithms are implemented in C++, running on a PC with a 2.4GHz Core 2 CPU, and only one core is used in the computation.

First of all, we test our algorithm DBS on all the line drawings of planar-faced manifolds appearing in previous papers about line drawing interpretation or 3D reconstruction from line drawings, such as [11], [16], [25], [13], [28], [8], [4], [15],and [9]. We find that DBS can efficiently identify the real faces from all the line drawings within several seconds for each. However, SSTS takes

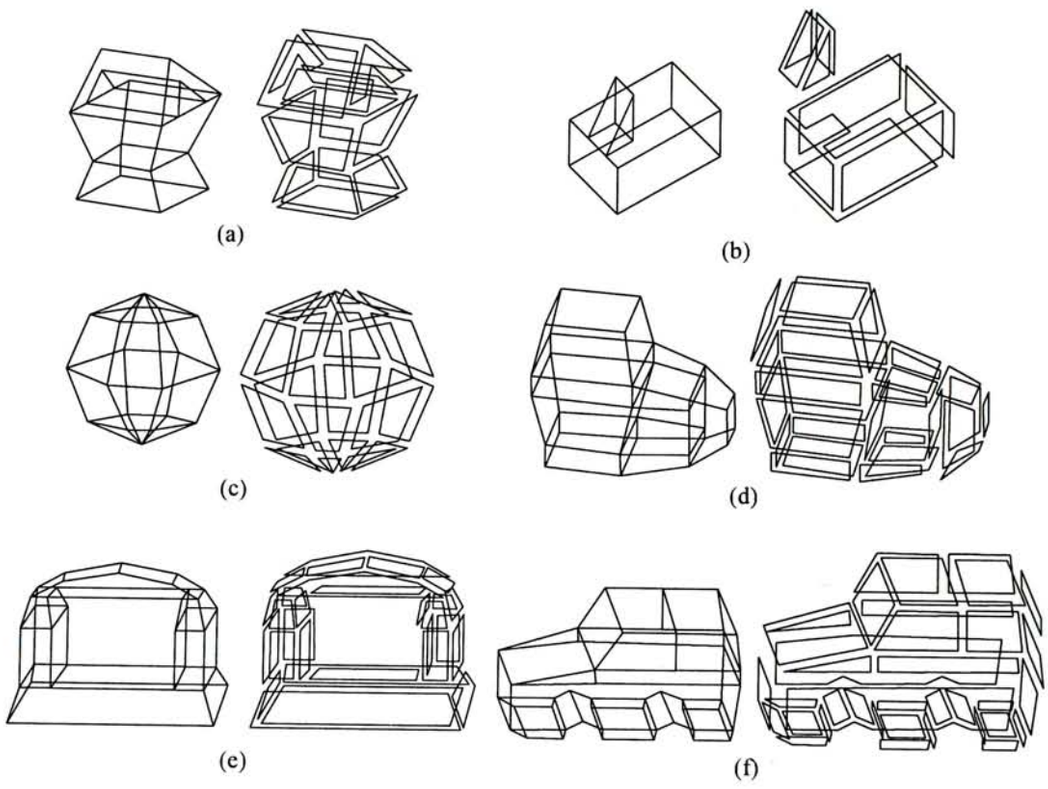


Figure 2.6: Six line drawings and their real faces found by our algorithm.

unacceptable times for many complex line drawings.

In Figure 2.6, six line drawings are given with their corresponding real face configurations. We test SSTS and DBS on these line drawings. Table 2.1 summarizes the results. In Table 2.1, the term 'PFCs in the Search' under 'SSTS' denotes the PFCs that are used as the input to the tree search step in SSTS; the term 'PFCs in the Search' under 'DBS' denotes the PFCs that are the faces and PFCs obtained by steps 1–3 in Algorithm 3, which are the input to the real face detection (step 4) in Algorithm 3. The table also lists the computational times of the two algorithms for the line drawings.

From Table 2.1, we can see that both SSTS and DBS are fast to find the real faces from line drawings (a) and (b). However, SSTS takes much more times to deal with line drawings (c) and (d). Even worse, SSTS cannot finishes its search in one day for line drawing (e) or (f). In contrast, DBS is fast enough

to handle every line drawing with about one second or less.

It is not difficult to explain why DBS is efficient but SSTS is not for a complex line drawing. From Table 2.1, we can see that there are many PFCs inputted to the tree search step in SSTS for line drawings (e) and (f), which are 1151 and 1787 PFCs, respectively, generating huge search trees. However, in DBS, all the four steps are fast and the numbers of PFCs inputted to steps 2, 3, and 4 are all small.

Figure 2.7 shows six more complex line drawings. DBS can identify all the real faces correctly from them. To save space, we do not show the real face configurations. Table 2.2 lists the computational times of DBS. We can see again that DBS is efficient enough for practical use. Note that SSTS cannot obtain the result in one day for each of the line drawings.

To find out how the computational time of DBS varies for line drawings of increasing size, 10 tower chains TC_{1-10} are used in this experiment. As shown in Figure 2.8(a), TC_{1-10} have 1–10 tower(s), respectively. The times taken by DBS are given in Figure 2.8(b). It can be seen that the time increases approximately linearly with the size of the tower chain. For the most complex tower chain with 264 faces and 536 edges, DBS needs only 42.6 seconds to identify all the real faces.

So far, we have not found an example in which DBS fails to identify the real faces. In our future work, we will try to find its limitation with more complex line drawings of manifolds.

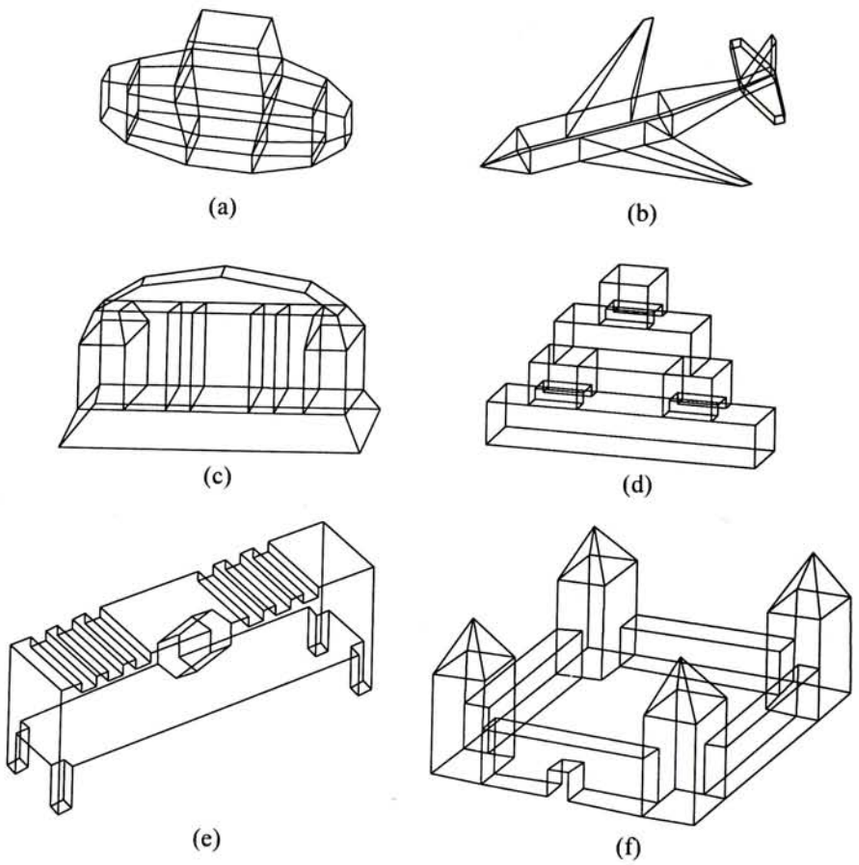


Figure 2.7: Six more complex line drawings.

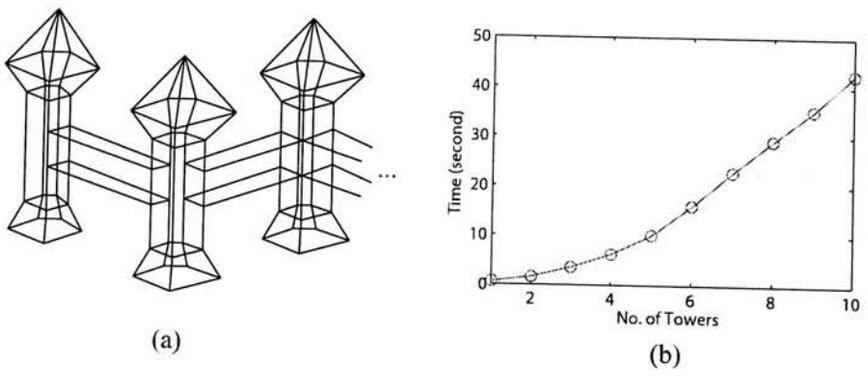


Figure 2.8: (a) The line drawing of a tower chain. (b) Times taken by DBS to find the real faces from these tower chains.

Table 2.1: Results obtained by SSTS and DBS from the line drawings in Figure 2.6.

	Real Faces	Edges	SSTS		DBS	
			PFCs	Time	PFCs	Time
(a)	16	36	37	0.125s	23	0.204s
(b)	10	22	12	0.109s	12	0.156s
(c)	24	42	202	289s	30	0.499s
(d)	30	62	460	1883s	39	0.748s
(e)	29	63	1151	>1 day	37	0.702s
(f)	37	81	1787	>1 day	42	1.02s

Table 2.2: Computational times taken by DBS to deal with the line drawings in Figure 2.7.

	(a)	(b)	(c)	(d)	(e)	(f)
Real Faces	42	38	41	36	46	56
Edges	88	75	95	106	138	124
Time	1.25s	0.921s	1.75s	1.05s	2.31s	1.22s

Chapter 3

3D Reconstruction

In this chapter, we propose an approach called object cut to tackle an important problem in computer vision, 3D object reconstruction from single line drawings. Given a complex line drawing representing a solid object, our algorithm finds the places, called cuts, to separate the line drawing into much simpler ones. The complex 3D object is obtained by first reconstructing the 3D objects from these simpler line drawings and then combining them together. Several propositions and criteria are presented for cut finding. A theorem is given to guarantee the existence and uniqueness of the separation of a line drawing along a cut. Our experiments show that the proposed approach can deal with more complex 3D object reconstruction than state-of-the-art methods.

3.1 Assumption and Terminology

In this paper, we consider the reconstruction of the same kind of objects as that in [17], i.e., planar-faced manifolds, which are the most common solids (see below for their definition). A line drawing, represented by a single edge-vertex graph, is the parallel or near parallel projection of the edges of a manifold in a generic view with all its edges and vertices visible. Same as [17], we also assume that the faces of a manifold are available from its line drawing. Finding faces

from a line drawing with hidden lines visible has been well studied in previous work [1], [9], [14], [15], [16], [25]. Next, we list the terms used in the rest of this paper, most of which are exemplified in Fig. 3.1.

- **Manifold**. A manifold, or more specifically 2-dimentional manifold, is a solid where every point on its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space. Manifolds considered in this paper are made up of flat faces. In this kind of manifolds, each edge is shared exactly by two faces [7].
- **Face**. A face is a flat patch of a manifold bounded by edges.
- **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 [17].
- **Degree of a vertex**. The number of edges adjacent to a vertex is called the degree of the vertex.
- **Artificial line**. An artificial line is a line used to indicate the coplanarity of two cycles [17].
- **Chord**. A chord of a cycle is an edge or a virtual line inside the cycle that connects two nonadjacent vertices of the cycle. A virtual line does not appear as an edge in the original line drawing.
- **Neighboring face**. If a face f_1 has a sharing edge with another face f_2, then f_2 is called a neighboring face of f_1. If an edge is on the boundary of a face, then the face is called a neighboring face of the edge.
- **Rotation direction of a face**. For a manifold consisting of planar faces, there exists an assignment of a rotation direction for each face simultaneously such that the rotation directions of any two neighboring faces on their sharing edge(s) are opposite [23].
- **Cut**. A cut is a planar cycle on the surface of a manifold, formed by

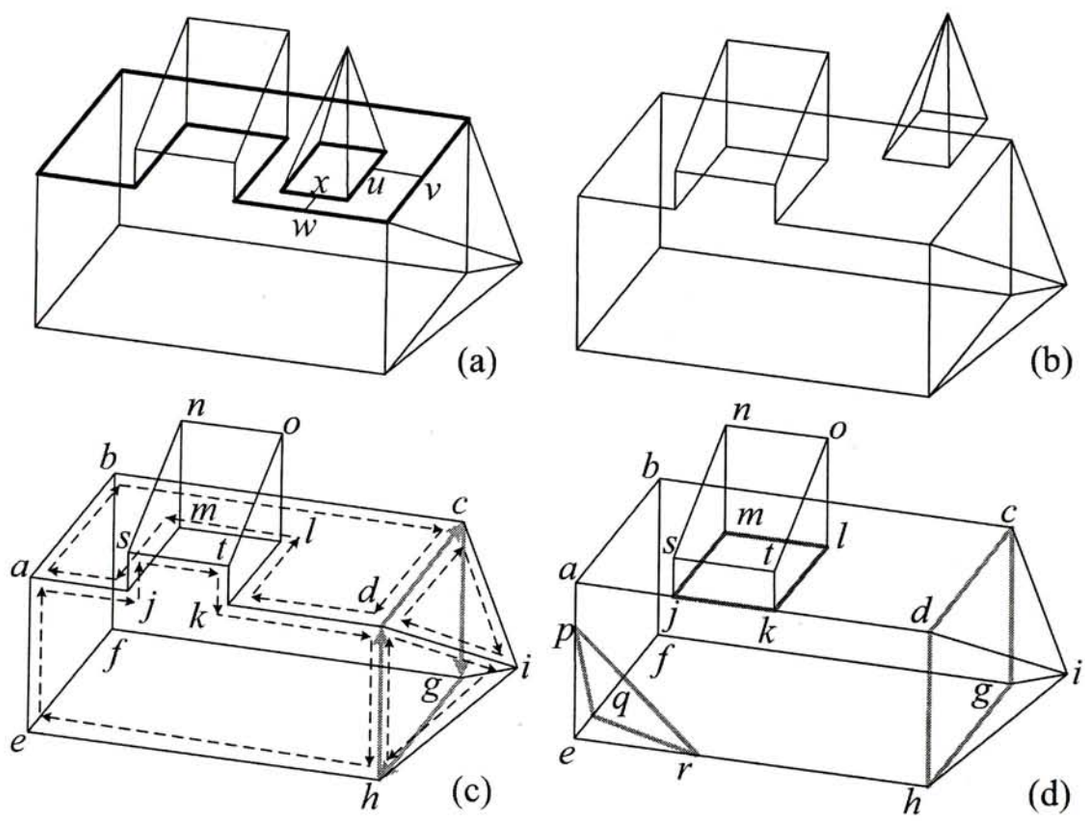


Figure 3.1: Examples of most of the terms. In Fig. 3.1(a), edges (u, v) and (w, x) are two artificial lines. Fig. 3.1(b) shows the resulted sub-manifolds after removing them. In Fig. 3.1(c), cycle $(a, j, s, t, k, d, h, e, a)$ is a face. Face (d, i, h, d) is a neighboring face of face (c, i, d, c). The dashed arrows show the rotation directions of four faces $(a, j, s, t, k, d, h, e, a)$, $(a, b, c, d, k, l, m, j, a)$, (c, i, d, c), and (d, i, h, d). Note that on edges (d, k), (a, j), (d, c), (d, i), and (d, h), the rotation directions of their two neighboring faces are opposite. The bold (red) arrows show the rotation direction of cut (c, g, h, d, c), which is arbitrarily assigned. Face $(a, j, s, t, k, d, h, e, a)$ is inconsistent with cut (c, g, h, d, c), but face (d, i, h, d) is consistent with it. Edge (e, f) is a chord of cycle (a, b, f, g, h, e, a), and a virtual line connecting vertices e and g is also a chord of the cycle. In Fig. 3.1(d), cycles (p, q, r, p), (c, g, h, d, c), and (m, l, k, j, m) are cuts. Face $(a, j, s, t, k, d, h, e, a)$ is a neighboring face of cut (c, g, h, d, c). Face (c, d, i, c) is connected to cuts (c, g, h, d, c). Edge (f, g) is connected to cut (c, g, h, d, c).

cutting the manifold with a plane, consisting of some edges of the manifold and/or new edges on the surface of the manifold, with its enclosed region not on the surface of the manifold. In this paper, we also assign a rotation direction to a cut.

- **Connected faces and edges of a cut**. If a face has at least one sharing point with a cut, then the face is called connected to the cut. If an edge has a sharing point with a cut and is not an edge of the cut, then the edge is called connected to the cut.
- **Neighboring face of a cut**. If a cut has at least one sharing edge with a face, the face is called a neighboring face of the cut.
- **Consistency of a face with a cut**. Given a cut and one of its neighboring faces, if they have the same rotation direction on their sharing edge(s), they are called consistent; otherwise, they are called inconsistent.
- **Partition of a set**. Given a non-empty set S, a partition of S is denoted by $P(S) = (S_0, S_1)$ where S_0 and S_1 are two non-empty sets with $S_0 \cup S_1 = S$ and $S_0 \cap S_1 = \phi$.

Artificial lines, added by the designer, are used to indicate the coplanarity of two cycles in solid modeling [1], [4], [15], [17]. One example is shown in Fig. 3.1(a) where two artificial lines (u, v) and (w, x) indicate that the two bold cycles are coplanar. Without them, it is impossible to know the geometric relation between the two objects in Fig. 3.1(b). Detecting artificial lines is an easy task according to the connection between an artificial line and the edges it connects to [17]. After removing the artificial lines in Fig. 3.1(a), the line drawing becomes two line drawings without artificial lines (Fig. 3.1(b)). In the next Sections 3.2 and 3.3, we consider line drawings without artificial lines.

3.2 Finding Cuts from a Line Drawing

The main difference between a cut and an internal face is that the former can have part or all of its boundary not from the original line drawing, while the latter consists of edges all from the original line drawing. An internal face is a special case of a cut. The strict requirement of internal faces makes the partition algorithm in [17] fail when a complex manifold is without, or with too few, internal faces (see Fig. 1.7 for example).

According to the definition of a cut, there is an infinite number of cuts on a manifold. We should find those cuts that really simplify the reconstruction problem. Internal faces are good cuts to partition a line drawing, such as the one in Fig. 1.7(b) and the cut (c, g, h, d, c) in Fig. 3.1(d). However, the cut (p, q, r, p) in Fig. 3.1(d) is not a good cut, and a cut that separates a rectangular solid into two rectangular solids is not a good cut either, because they do not simplify the reconstruction. In the next two sub-sections, we present some propositions and criteria for finding good cuts.

3.2.1 Propositions for Finding Cuts

Given a cycle on the surface of a manifold, determining whether the cycle is a cut is not a trivial problem due to the lack of 3D geometry in a 2D line drawing. Here we present three propositions to eliminate cycles that cannot be cuts.

Property 5. *A cycle is not a cut if it is self-intersecting.*

When we cut a manifold with a plane to form a cut, the cut becomes two visible planar faces, which are not self-intersecting obviously.

Property 6. *A cycle is not a cut if it has a chord inside it and the chord is on the surface of the manifold.*

Proof. As shown in Fig. 3.2, suppose that the cycle $(\ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots$ is a cut with a chord (v_i, v_j). Let v be a point in the middle of the chord. If

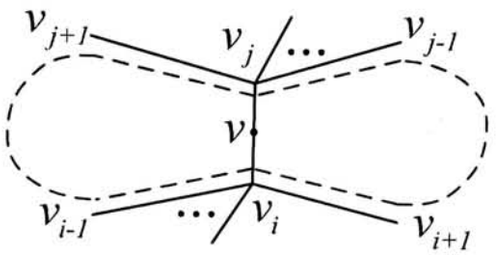


Figure 3.2: Part of a line drawing where the cycle has a chord (v_i, v_j).

the chord is an edge of the original line drawing, then v inside the cut is on the surface of the manifold. If the chord is not an original edge but is on a face, then v inside the cut is still on the surface of the manifold. Both cases contradict the definition of a cut. □

Property 7. *A cycle is not a cut if it has two non-collinear edges belonging to a face and there is an overlapping region between it and the face in the 2D line drawing plane.*

Proof. Since both a cut and a face are planar, if the cut has two non-collinear edges on the face, they must be coplanar. Furthermore, when they have an overlapping region in the 2D line drawing plane, they overlap in the 3D space. Thus, this region inside the cut is on the surface of the manifold, which contradicts the definition of a cut. □

Note that Proposition 7 implies that a face is not a cut.

3.2.2 Searching for Good Cuts

Through observation of common manifold objects, we find that good cuts that separate a complex manifold into simpler ones usually follow the following three criteria:

Criterion 1. *A good cut should have as few new edges as possible.*

Criterion 2. *A good cut should have as few edges as possible.*

Criterion 3. *A new edge added to form a good cut is usually parallel (or near parallel) to an original edge of the face which the new edge is on.*

We have Criterion 1 because adding too many new edges into the line drawing introduces many new faces on the surface of the object, making the reconstruction more complicated. It also gives priority to finding internal faces. Criterion 2 comes from the observation that a cut with too many edges may lead to an untidy partition of the line drawing. Criterion 3 is based on the fact that a large man-made object is often formed by regular/symmetric smaller objects.

With the criteria and the propositions, we next develop a shortest cycle algorithm to search for good cuts on a search graph. A search graph is constructed based on a line drawing where new edges are added connecting vertices in each face. Fig. 3.3 shows an example where the dashed lines are part of the new edges. Note that a new edge connecting two vertices on different faces is not added because it is not on the surface of the object. Besides, we do not add new vertices to make the problem too complicated since our current method can already obtain excellent results that are in accordance with our visual perception of the object partition (see the experiments).

In a search graph, each edge has a weight defined by

$$\omega(e) = \begin{cases} 1, & \text{if } e \text{ is from the original line drawing,} \\ \alpha \cdot (\min_{e' \in S(e)} \gamma(e, e')) + \beta, & \text{otherwise,} \end{cases} \tag{3.1}$$

where e is an original or new edge of the search graph, α and β are two parameters used to balance the criteria, and we set $\alpha = \beta = 3$ in this paper. $S(e)$ is an edge set consisting of all the edges of the face on which the new edge e lies, and $\gamma(e, e')$ is a function evaluating the parallelism between two edges e and e' defined as:

$$\gamma(e, e') = 1 - \exp(-0.05\theta_{e,e'}^2), \tag{3.2}$$

where $\theta_{e,e'} \in [0, 90]$ is the angle between e and e'. Take three edges, $e_1 = (a, b)$, $e_2 = (j, k)$, and $e_3 = (k, h)$ in Fig. 3.3, for example. We have $\omega(e_1) = 1$,

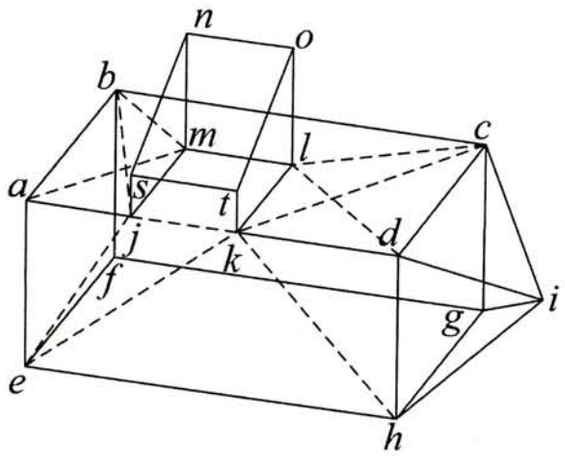


Figure 3.3: A searching graph where only part of the new edges (dashed lines) are shown for clarity.

$\omega(e_2) \approx 3$, and $\omega(e_3) \approx 6$. Given an original edge of the search graph, the shortest cycle passing through this edge is the one that has the minimum sum of the weights of the edges on this cycle. However, not every shortest cycle can be a cut. The three propositions given in Section 3.2.1 help to eliminate cycles that cannot be a cut. For example, the shortest cycle passing through edge (c, d) in Fig. 3.3 is (c, d, i, c), but it cannot be a cut according to Proposition 7 (it is a face actually).

Algorithm 4 lists the steps to find a best cut from a line drawing. In the algorithm, $\omega(\mathfrak{c})$ denotes the total weight of a cut $\mathfrak{c}$. In step 5, the returned shortest path p connected with edge e_i forms a cut, and $\mathfrak{c}$ is always the shortest cut found so far. The procedure *ExtDijkstra* is an extended version of Dijkstra's shortest path algorithm for finding a shortest path on a weighted graph. The main difference between them is that Propositions 5–7 are incorporated into the search in *ExtDijkstra*. Proposition 5 is used earlier than Propositions 6 and 7 so that most of the paths that cannot be a cut are determined as soon as possible. Since the conditions in Propositions 6 and 7 need to be tested when a cycle is formed, they are used later. Note that in step 8, each path in $PathSet(v_{i,2})$ connected with edge $(v_{i,1}, v_{i,2})$ forms a cycle. Another difference

between *ExtDijkstra* and Dijkstra's algorithm is that we need to keep the geometric positions of the vertices and edges of G' so that Propositions 5–7 are used correctly.

Only one cut is obtained by each running of Algorithm 4. When the current best cut is found, we use the partition method proposed in the next section to separate the line drawing along this cut. Then Algorithm 4 is run again on each separated line drawing. This procedure is repeated until some condition is satisfied. The stopping condition we use is

$$\mu = \frac{\text{sum of the edge weights of the best cut}}{\text{number of faces of the line drawing}} > 1 \ . \tag{3.3}$$

The ratio μ is large when the line drawing is simple (i.e., with fewer faces) and the cut is not good (i.e., with a large weight).

3.3 Separation of a Line Drawing from Cuts

To separate a manifold into simpler ones, cuts of the manifold are found first and then the manifold is separated from the cuts. In Fig. 3.1(d), some cuts separate faces of the original manifold into sub-faces, which generates a new representation of the manifold. We call this representation an extended manifold, which is defined below.

Definition 3.1. *Let C be a set of cuts. Let F, E and V be the face set, edge set, and vertex set of the original manifold, respectively. The extended manifold is an object with its face set $F_{ext}(C)$, edge set $E_{ext}(C)$, and vertex set $V_{ext}(C)$ defined as follows: 1) For each face $f \in F$, if C separates f into two or more sub-faces $\{f_i\}$, then $\{f_i\} \subset F_{ext}(C)$; otherwise, $f \in F_{ext}(C)$. Besides, $F_{ext}(C)$ contains no other faces not from these two cases. 2) $E_{ext}(C) = E \cup \{$edges of the cuts in $C\}$. 3) $V_{ext}(C) = V$.*

Since we do not consider cuts with new vertices, $V_{ext}(C)$ is the same as V. It is easy to see that an extended manifold is still a manifold, because all

Algorithm 4 Finding a best cut

Input: A Line Drawing $G = (V, E)$
Initialization: The best cut $\mathfrak{c} \leftarrow null$; $\omega(\mathfrak{c}) \leftarrow \infty$

1. Create the searching graph $G' = (V, E')$ from G
2. Calculate the weight of each edge of E'
3. **for** each edge $e_i = (v_{i,1}, v_{i,2}) \in E$
4. Call $ExtDijkstra(v_{i,1}, v_{i,2})$
5. **if** a shortest path p is found and $\omega(p, e_i) < \omega(\mathfrak{c})$,
   then $\mathfrak{c} \leftarrow (p, e_i)$

Output: The best cut $\mathfrak{c}$
procedure $ExtDijkstra(v_{i,1}, v_{i,2})$

1. Remove edge $(v_{i,1}, v_{i,2})$ from E'
2. $d(v) \leftarrow \infty$, $PathSet(v) \leftarrow \phi$, $\forall v \in V$;
   $d(v_{i,1}) \leftarrow 0$; $PathSet(v_{i,1}) \leftarrow \{(v_{i,1})\}$; $S \leftarrow \phi$, $Q \leftarrow V$
3. **while** $v_{i,2} \notin S$
4. Move vertex $u \in Q$ with minimum $d(u)$ from Q to S
5. **for** each edge $(u, v) \in E'$ with $v \in Q$
6. Extend the paths in $PathSet(u)$ by (u, v); add them to $PathSet(v)$; remove the paths from $PathSet(v)$ that cannot form a cut according to Proposition 5
7. **if** $PathSet(v) = \phi$, **then** $d(v) \leftarrow \infty$;
   else $d(v) \leftarrow \min_{p \in PathSet(v)} \omega(p)$
8. Remove the paths from $PathSet(v_{i,2})$ that cannot form a cut when they are connected with edge $(v_{i,1}, v_{i,2})$ according to Propositions 6 and 7
9. Add edge $(v_{i,1}, v_{i,2})$ back to E'; **if** $PathSet(v_{i,2}) = \phi$, **return** $null$; **else return** the shortest path p in $PathSet(v_{i,2})$

end of $ExtDijkstra(v_{i,1}, v_{i,2})$

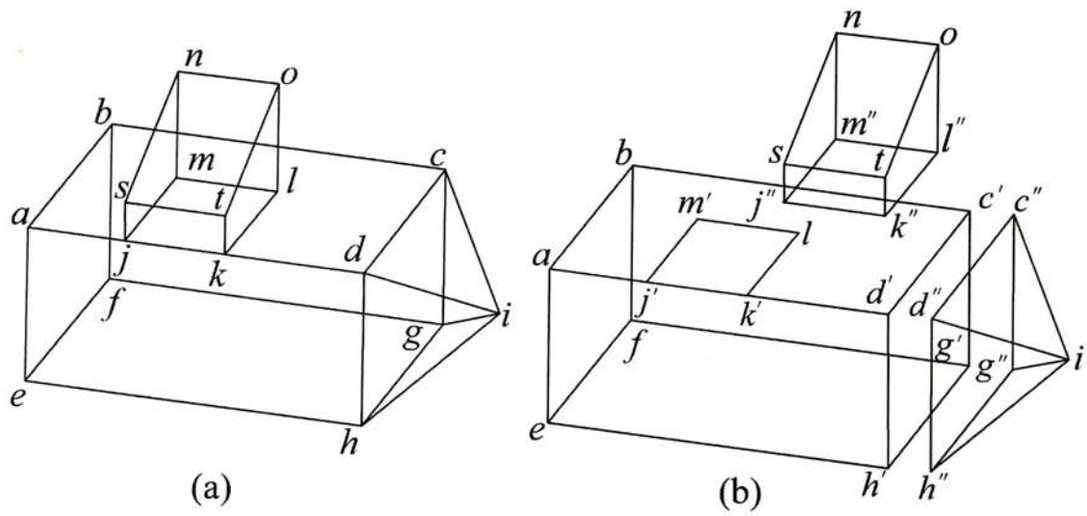


Figure 3.4: (a) An extended manifold. (b) Three separated manifolds.

the new edges are on the surface of the original manifold and each edge of the extended manifold is still passed through by exactly two faces of the extended manifold. Some faces of the original manifold are broken into sub-faces by the cuts. For example, Fig. 3.4(a) is an extended manifold from the manifold in Fig. 3.1(c) with a cut set $C = \{(c, g, h, d, c), (m, l, k, j, m)\}$. Newly generated faces (a, e, h, d, k, j, a) and (s, j, k, t, s) are the result of cut (m, l, k, j, m). Note that cut (c, g, h, d, c) does not generate any new faces.

Separating an extended manifold along a cut in a 2D line drawing is not trivial. The difficulties include the lack of 3D geometry in a line drawing and the fact that the faces and edges connected to a cut can appear in any directions with respect to the cut. Besides, when the line drawing is separated into two sides along a cut, it is not obvious which side an edge connected to the cut should be put in.

Even though there exist these difficulties, humans are quite easy to obtain a unique separation along a cut. For example, given the extended manifold shown in Fig. 3.4(a) with the two cuts (c, g, h, d, c) and (m, l, k, j, m), humans always generate the separations in Fig. 3.4(b). We can see that the faces connected to a cut are separated into two non-empty sets, each of which contains

the faces on one side. Besides, for each edge connected to the cut, its two neighboring faces always appear on one side only. This is because if the two neighboring faces appear on different sides, the line drawing cannot be separated into two sides along the cut. Next, a formal definition of the separation is given, which is called *a partition along a cut.*

Definition 3.2. *Given a cut* $\mathfrak{c}$*, let the set of all the faces connected to* $\mathfrak{c}$ *be* $F(\mathfrak{c})$*, and let the set of all the edges connected to* $\mathfrak{c}$ *be* $E(\mathfrak{c})$*. A partition of the extended manifold along* $\mathfrak{c}$ *is to find a face set partition* $P(F(\mathfrak{c})) = (F_0(\mathfrak{c}), F_1(\mathfrak{c}))$ *and an edge set partition* $P(E(\mathfrak{c})) = (E_0(\mathfrak{c}), E_1(\mathfrak{c}))$ *simultaneously such that for any edge* $e \in E_m(\mathfrak{c})$*, it holds that* $e \notin Edge(f)$*,* $\forall f \in F_{1-m}(\mathfrak{c})$*,* $m = 0, 1$*, where* $Edge(f)$ *denotes the set of all the edges of face* f*.*

For example, given the cut $\mathfrak{c} = (m, l, k, j, m)$ in Fig. 3.4(a), the face set partition along $\mathfrak{c}$ is: $F_0(\mathfrak{c}) = \{(a, e, h, d, k, j, a), (a, b, c, d, k, l, m, j, a)\}$ and $F_1(\mathfrak{c}) = \{(s, j, m, n, s), (s, j, k, t, s), (t, k, l, o, t), (n, m, l, o, n)\}$, and the edge set partition along $\mathfrak{c}$ is: $E_0(\mathfrak{c}) = \{(a, j), (k, d)\}$, $E_1(\mathfrak{c}) = \{(s, j), (n, m), (o, l), (t, k)\}$. Next we give a theorem showing that the partition along a cut exists and is unique[1] .

Theorem 3.3. *The partition of an extended manifold along a cut exists and is unique.*

Proof. Recall the rotation direction of a face. We can assign a rotation direction to every face of the manifold[2] such that any two neighboring faces have opposite rotation directions on their sharing edge(s). Let $\mathfrak{c} = (v_1, v_2, v_3, \ldots, v_n, v_1)$ be a cut with n vertices, and N_i be a neighborhood around v_i on the surface of the manifold such that N_i is topologically equivalent to a 2D open disk and small enough with only the edges connected to v_i contained in N_i. According

[1] In [17], a theorem is given to show that the partition along an internal face exists and is unique. Theorem 3.3 in this paper is an extension of it.

[2] In this proof, the term *manifold* is used to denote the extended manifold for conciseness.

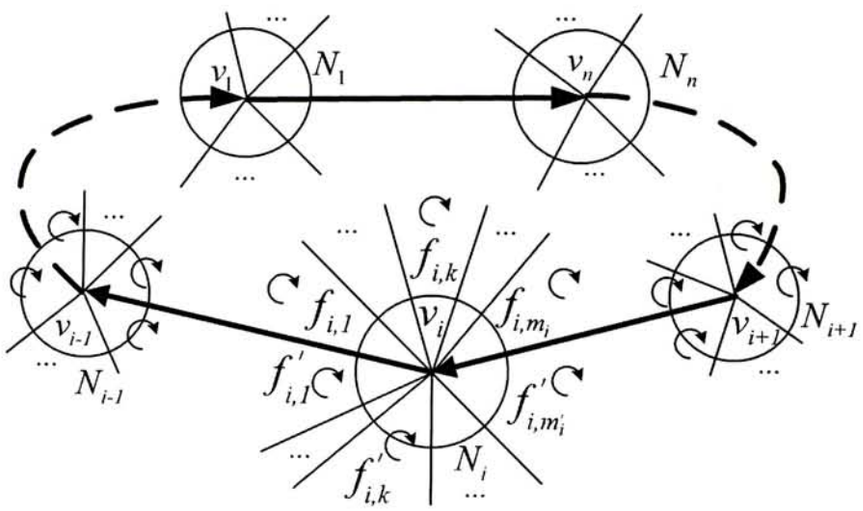


Figure 3.5: Part of a line drawing where a cut with n vertices is shown in bold lines, and all the neighborhoods $N_i, i = 1, 2, \ldots, n$, are stretched into 2D open disks.

to the definition of a manifold, every N_i can be stretched into a 2D open disk where the faces passing through v_i are located side by side around v_i without overlap. Besides, we arbitrarily assign a rotation direction to the cut, as shown in Fig. 3.5, which is helpful to create the face set partition and the edge set partition. Next we state and verify five properties when all the N_i's are stretched into 2D disks.

1) In N_i, two edges (v_{i-1}, v_i) and (v_i, v_{i+1}) of the cut separate the faces passing through v_i into two non-empty sets: $\{f_{i,1}, f_{i,2}, \ldots, f_{i,m_i}\}$ on one side and $\{f'_{i,1}, f'_{i,2}, \ldots, f'_{i,m'_i}\}$ on the other side (see Fig. 3.5). Neither set can be empty because otherwise, N_i is not topologically equivalent to a 2D open disk.

2) All the rotation directions of the faces in N_i are either clockwise or counter-clockwise. Without loss of generality, suppose that the rotation direction of $f_{i,1}$ is clockwise (Fig. 3.5). Since two faces have opposite rotation directions on their sharing edge(s), the rotation direction of $f_{i,2}$ is also clockwise, and so are the rotation directions of $f_{i,3}, f_{i,4}, \ldots, f_{i,m_i}$. Similarly, all the rotation directions of $f'_{i,1}, f'_{i,2}, \ldots, f'_{i,m'_i}$ are also clockwise.

3) Among the four faces, $f_{i,1}$, f_{i,m_i}, $f'_{i,1}$, f'_{i,m'_i}, which pass through the cut and v_i, two of them lying on one side of the cut are consistent with the cut, while the other two lying on the other side are inconsistent with the cut[3] . In Fig. 3.5, $f_{i,1}$ and f_{i,m_i} lying on the upper side of the cut are consistent with the cut, while $f'_{i,1}$, f'_{i,m'_i} lying on the lower side are inconsistent with the cut. This property is a direct result of property 2).

4) Suppose that $f_{i,1}$ and f_{i,m_i} are consistent with the cut, and $f'_{i,1}$ and f'_{i,m'_i} are inconsistent with the cut, for $i \in \{1, 2, \ldots, n\}$. To create a face set partition, let $F_0(\mathfrak{c}) = \bigcup_{i=1}^{n} \{f_{i,1}, f_{i,2}, \ldots, f_{i,m_i}\}$ and $F_1(\mathfrak{c}) = \bigcup_{i=1}^{n} \{f'_{i,1}, f'_{i,2}, \ldots, f'_{i,m'_i}\}$. That is, $F_0(\mathfrak{c})$ contains the faces consistent with the cut and the faces lying on the same side as these consistent faces, and $F_1(\mathfrak{c})$ contains all other faces passing through v_1, v_2, ..., or v_n. To create an edge partition, let $E_0(\mathfrak{c})$ be the set of edges whose neighboring faces belong to $F_0(\mathfrak{c})$, and $E_1(\mathfrak{c})$ be the set of edges whose neighboring faces belong to $F_1(\mathfrak{c})$. Since for any $e \in E(\mathfrak{c})$, the two neighboring faces of e lie on the same side of the cut, they must belong to the same face partition set, i.e., either $F_0(\mathfrak{c})$ or $F_1(\mathfrak{c})$. Therefore $P(F(\mathfrak{c})) = (F_0(\mathfrak{c}), F_1(\mathfrak{c}))$ and $P(E(\mathfrak{c})) = (E_0(\mathfrak{c}), E_1(\mathfrak{c}))$ form a partition along $\mathfrak{c}$ that satisfies Definition 3.2. This property and the property 1) show the existence of a partition along $\mathfrak{c}$.

5) The above partition along $\mathfrak{c}$ is unique. To verify this property, suppose that there is another face set partition $(F'_0(\mathfrak{c}), F'_1(\mathfrak{c}))$ along $\mathfrak{c}$ with $F'_0(\mathfrak{c}) \neq F_0(\mathfrak{c})$ (and thus $F'_1(\mathfrak{c}) \neq F_1(\mathfrak{c})$). Since none of $F'_0(\mathfrak{c})$ and $F'_1(\mathfrak{c})$ is empty, there must exist an edge $e \in E(\mathfrak{c})$, the two neighboring faces of which belong to different face partition sets. In this case, the line drawing cannot be separated into two sides because e appears in both sides, which shows that $(F'_0(\mathfrak{c}), F'_1(\mathfrak{c}))$ is not a valid face set partition. □

Theorem 3.3 and Definition 3.2 already provide a method to partition a line

[3]It is possible that there is only one face on one side. In this case, the two faces merge into one.

Algorithm 5 Partition of a line drawing along a cut $\mathfrak{c}$

1. Create the extended manifold with $\mathfrak{c}$
2. $E_0(\mathfrak{c}) \leftarrow \phi$; $E_1(\mathfrak{c}) \leftarrow \phi$; $F_0(\mathfrak{c}) \leftarrow \phi$;
   $F_1(\mathfrak{c}) \leftarrow$ {faces connected to $\mathfrak{c}$ in the extended manifold}
3. Pick a face $f \in F_1(\mathfrak{c})$ and move it to $F_0(\mathfrak{c})$
4. **for** each face $f \in F_1(\mathfrak{c})$
5. **if** f shares an edge connected to $\mathfrak{c}$ with a face in $F_0(\mathfrak{c})$, **then** move f to $F_0(\mathfrak{c})$
6. **for** each edge e connected to $\mathfrak{c}$
7. **if** e is on a face in $F_m(\mathfrak{c})$, $m = 0$ or 1, **then** $E_m(\mathfrak{c}) \leftarrow e$

drawing along a cut, which is given in Algorithm 5. After the partition along $\mathfrak{c}$, $\mathfrak{c}$ becomes two new faces on the two sides of the partition. In Fig. 3.4(b), three objects are generated from the partitions along the two cuts (m, l, k, j, m) and (c, g, h, d, c) in Fig. 3.4(a). Note that the newly face (m', l', k', j', m') is merged with the original face $(a, j', m', l', k', d, c, b, a)$ in the biggest object in Fig. 3.4(b) by deleting the edges connected to the vertices (m' and l') of degree 2. The following theorem shows that separated line drawings still represent manifolds.

Theorem 3.4. *After the partition along a cut, the line drawing (line drawings) still represents (represent) a manifold (manifolds).*

Proof. After the partition[4] , the new line drawing (line drawings) is (are) formed by the faces of the extended manifold and the two new faces from the cut. We only need to verify that every point on the new faces and their edges has a neighborhood topologically equivalent to a 2D open disk. Obviously, every point inside each new face satisfies this requirement. Let p be a point on an edge of a new face (the cut), say, at the middle of edge (v_i, v_{i-1}) in Fig. 3.5 without loss of generality. It is easy to find such a neighborhood around p, which is formed by points in $f_{i,1}$ or $f'_{i,1}$, edge (v_i, v_{i-1}), and the new face. □

[4]Note that one partition may or may not separate a line drawing into two disjoint line drawings.

Algorithm 6 3D reconstruction

1. Detect and delete all artificial lines
2. For each separated line drawing, find its best cut and partition it along the cut; repeat this step until $\mu > 1$
3. Reconstruct 3D manifolds from the separated line drawings
4. Combine the 3D manifolds to obtain a complete object

3.4 3D Reconstruction from a Line Drawing

After partitioning a line drawing along its cuts, we reconstruct 3D manifolds from these separated line drawings and then obtain the complete large 3D object through the combination of these smaller 3D manifolds.

It is not difficult to deal with 3D reconstruction from a separated line drawing because it is simple enough (see the experiments). We use the method in [13] to carry out this work. The basic idea of this method is to derive the z-coordinates of all the vertices by minimizing an objective function. Since a line drawing is considered as a parallel projection of a manifold and its face topology is known, the 3D object is obtained when all the z-coordinates are derived. More details can be found from [13]. After constructing the smaller 3D objects from all the separated line drawings, we merge them together to have a complex large object using the method in [17].

Algorithm 6 lists our complete algorithm to do 3D reconstruction from a complex line drawing.

3.5 Experiments

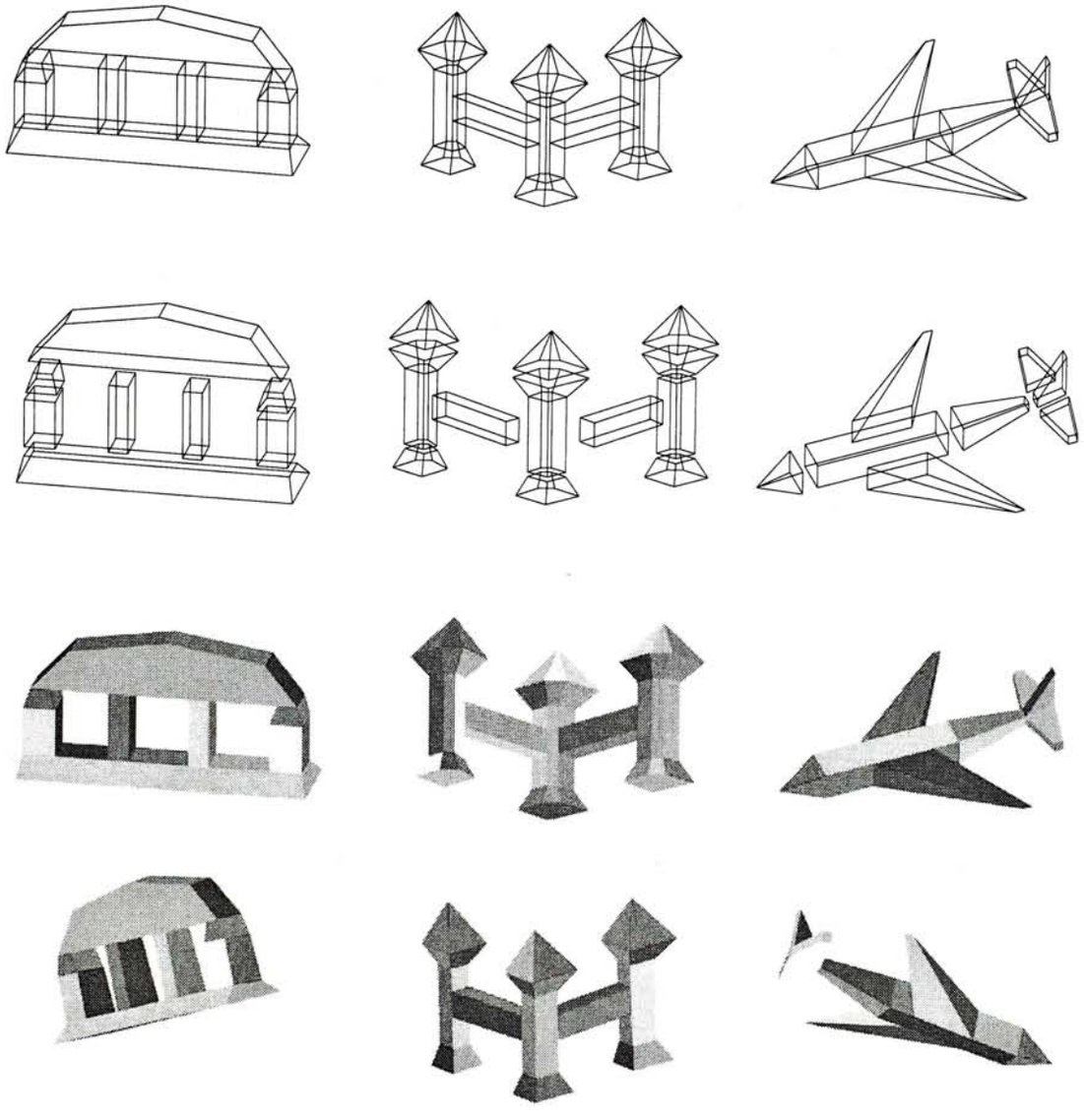
A set of examples is given in this section to demonstrate the performance of our algorithm. The problem in [17] is that it may fail when a complex line drawing has too few internal faces. For example, it can only separate the line

drawing in Fig. 1.7(a) into the two line drawings in Fig. 1.7(c). The larger line drawing in Fig. 1.7(c) is still too complex.

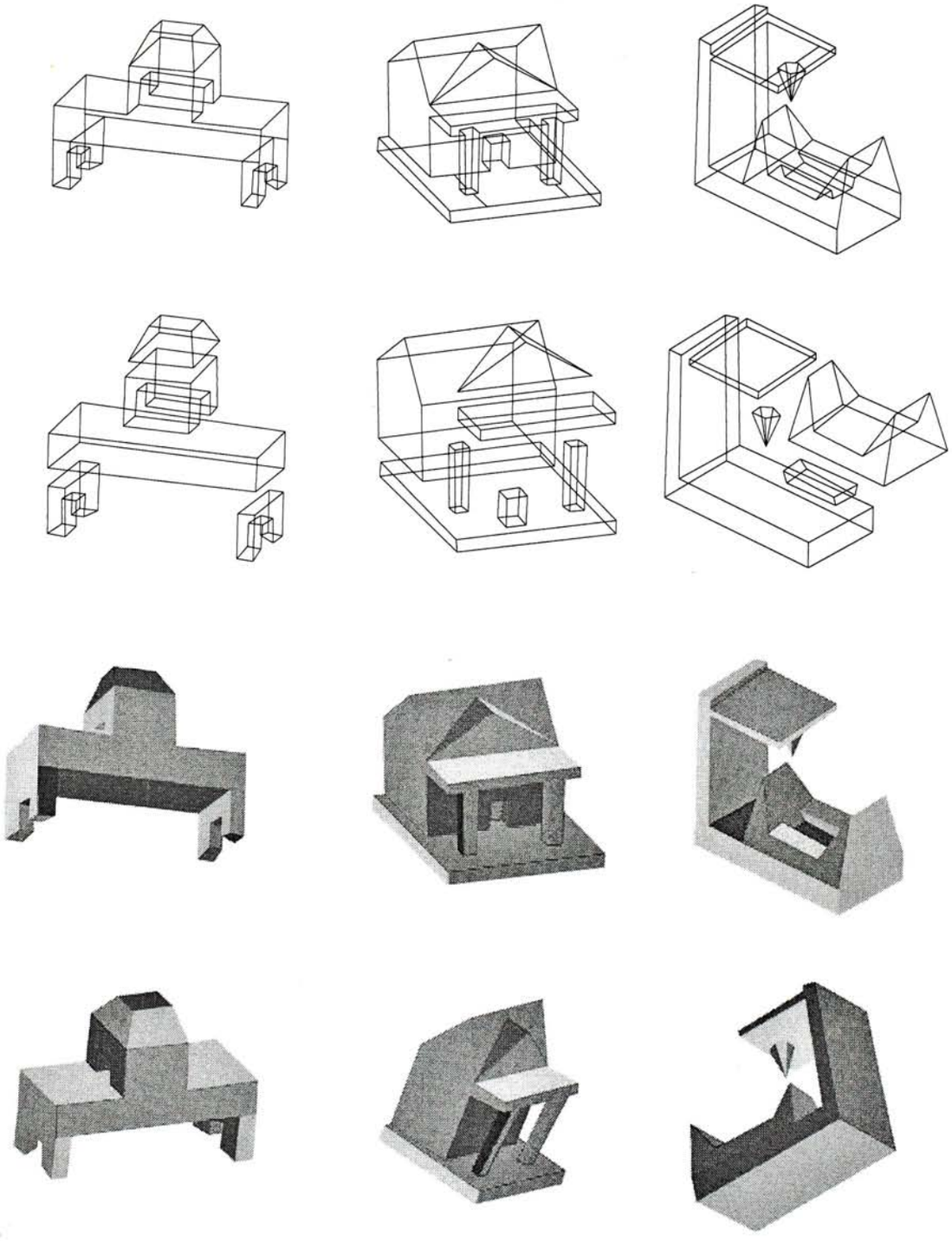
Fig. 3.6 presents a number of complex line drawings together with their partition and reconstruction results by our algorithm. From the second column of Fig. 3.6, we can see that our algorithm successfully finds good cuts to separate the line drawings, which are in accordance with our visual partitions. For the two objects in Figs. 3.6(a) and (b), our algorithm and the one in [17] obtain the same partition results. Note that besides the separations from the artificial lines, there is only one internal face in line drawing (d) or (e), and no internal face in line drawing (f), (g), or (h).

Because our algorithm can separate the complex line drawings into very simple line drawings based on the found cuts, the 3D reconstruction from these line drawings becomes much easier. From the third and forth columns in Fig. 3.6, we can see that the 3D objects are reconstructed very well. Besides, all the line drawings given in [17] can be dealt with by our algorithm because internal faces are special cases of cuts.

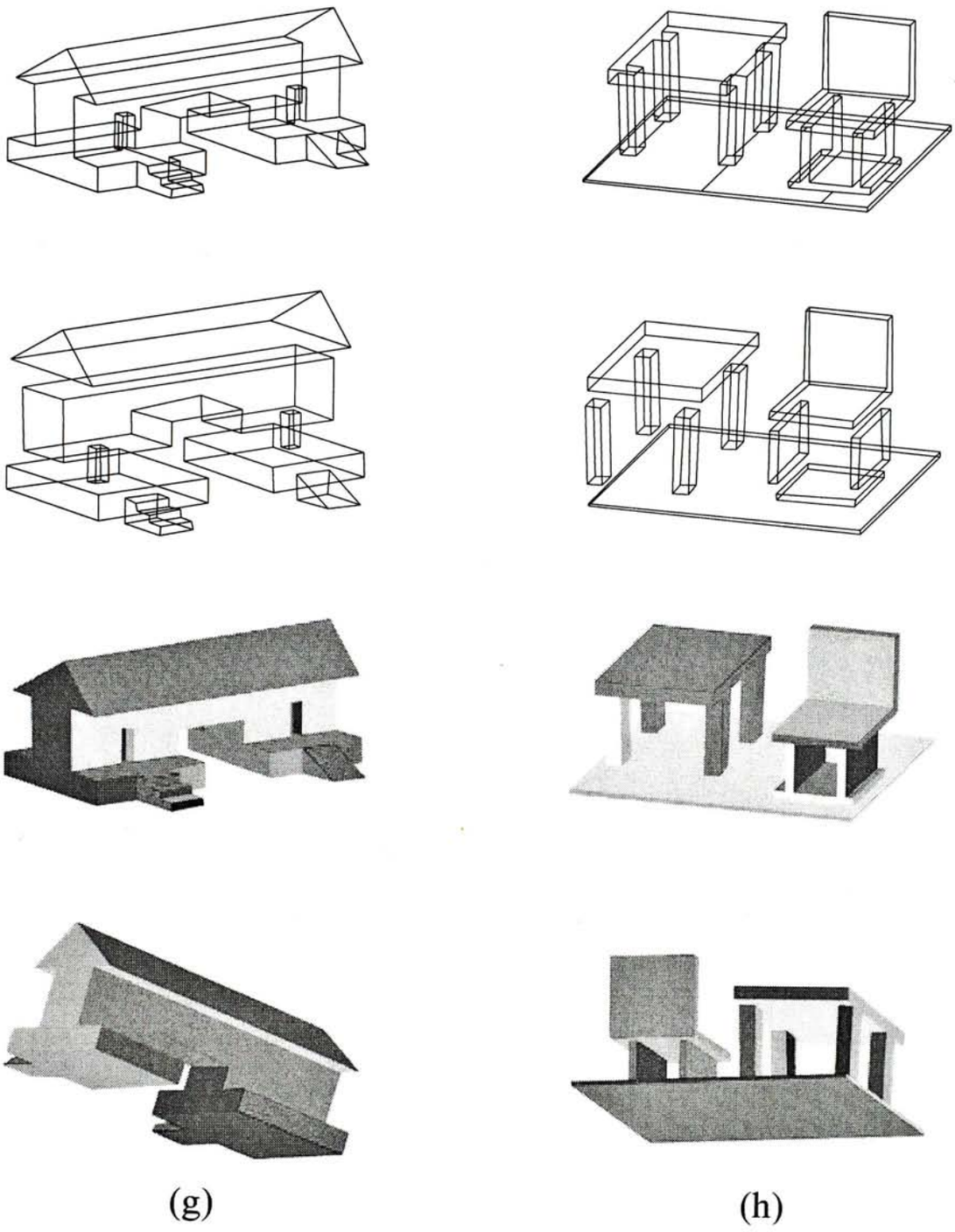
The computational time of Algorithm 6 depends on the complexity of a line drawing. It ranges from 10 to 112 seconds for the line drawings in Fig. 3.6. The algorithm is implemented using C++ and runs on a PC with 2.4GHz Intel Core2 CPU. Steps 3 and 4 consume the majority of the time, while steps 1 and 2 take about 1 second only for each of the line drawings.



(a) (b) (c)



(d) (e) (f)



(g) (h)

Figure 3.6: Experimental results on a set of complex line drawings (a)–(h) by our algorithm. The second rows shows the partitions of the line drawings. Each reconstructed 3D object is displayed in two views with its faces illustrated by different colors.

Chapter 4

Conclusion

In this thesis we develop a set of algorithms to recover the 3D geometry from a complex 2D line drawing. A 2D line drawing is one of most straight forward way to represent a 3D object. Recovering 3D geometry from a 2D line drawing is one of traditional topic in computer vision. The applications of this research include: flexible sketching interface for 3D model designers, converting existing industrial wireframe models to solid 3D models, 3D object generation from images with users' sketch, and 3D query creation for 3D object retrieval.

Previous approaches to recover 3D geometry from a 2D line drawing usually contains two steps: face configuration identification and 3D reconstruction. Although previous methods can correctly recover the 3D geometry for a simple 2D line drawing, when the line drawing becomes complex, most of previous methods fail either because that they have a very high computational cost, or because that the optimization algorithms are easily trapped in a local minimum. In order to solve this problem, we propose an efficient algorithm for the face configuration identification which has very low computational cost and a new 3D reconstruction algorithm which can avoid the local minimum problem when the line drawing become complex.

In face configuration identification part, we propose an efficient algorithm for the face identification from line drawings of manifolds. The first step of our algorithm is to find an initial face set using a previous fast algorithm for

the face identification from line drawings of sheet objects. Since this initial face set may contain undesirable internal faces and loses some real faces, the second and the third steps of our algorithm find potential lost faces. From all the outputs by these three steps, the last step of our algorithm detects the real faces and removes the others. Several geometric properties and a theorem have been presented for the design of our algorithm. Extensive experiments have been done to verify the performance of the algorithm, which is much more efficient to deal with a complex line drawing than previous ones.

In the 3D reconstruction part, we propose to separate a complex line drawing from cuts, which include internal faces as a special case. We develop several propositions and a criteria for cut finding. We also present a theorem that guarantees the existence and uniqueness of the partition of a line drawing along a cut. Our algorithm can tackle 3D reconstruction for more complex solid objects than previous algorithms.

In our future work, we will try to find the limitation of our face identification algorithm with more complex line drawings of manifolds, although currently we have not found an example in which our algorithm fails to identify the real faces. We will also try to extend our algorithms for more general objects such as objects with curved surfaces.

Bibliography

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

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

[3] 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. PAMI*, 30(3):507–517, 2008.

[4] Y. Chen, J. Liu, and X. Tang. A divide-and-conquer approach to 3D object reconstruction from line drawings. *ICCV*, 2007.

[5] M. Cooper. *Line Drawing Interpretation.* Springer, 2008.

[6] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. *ACM SIGGRAPH*, pages 11–20, 1996.

[7] D. E. LaCourse. *Handbook of Solid Modeling.* McGraw-Hill, Inc., 1995.

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

[9] H. Li, Q. Wang, L. Zhao, Y. Chen, and L. Huang. nD object representation and detection from single 2D line drawing. *LNCS*, 3519:363–382, 2005.

[10] Z. Li, J. Liu, and X. Tang. A Closed-form Solution to 3D Reconstruction of Piecewise Planar Objects from Single Images. *CVPR*, pages 1–6, 2007.

[11] 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.

[12] H. Lipson and M. Shpitalni. Correlation-based reconstruction of a 3d object from a single freehand sketch. *ACM SIGGRAPH*, 2007.

[13] J. Liu, L. Cao, Z. Li, and X. Tang. Plane-Based Optimization for 3D Object Reconstruction from Single Line Drawings. *IEEE Trans. PAMI*, 30(2):315–327, 2008.

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

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

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

[17] J. Liu and X. Tang. Decomposition of complex line drawings with hidden lines for 3D planar-faced manifold object reconstruction. *IEEE Trans. PAMI*, pages 3–15, 2010.

[18] R. E. M., J. Nievergelt, and N. Deo. *Combinatorial algorithms: theory and practices.* New Jersey: Prentice-Hall, 1977.

[19] T. Marill. Emulating the human interpretation of line-drawings as three-dimensional objects. *IJCV*, 6(2):147–161, 1991.

[20] P. Min, J. Chen, and T. Funkhouser. A 2D sketch interface for a 3D model search engine. *ACM SIGGRAPH Technical Sketches*, page 138, 2002.

[21] A. Piquer, R. Martin, et al. Using skewed mirror symmetry for optimisation-based 3d line-drawing recognition. In *In Proc. 5th IAPR International Workshop on Graphics Recognition (2003.* Citeseer, 2003.

[22] L. Ros and F. Thomas. Overcoming superstrictness in line drawing interpretation. *IEEE Trans. PAMI*, pages 456–466, 2002.

[23] H. Seifert. *Seifert and Threlfall: A Textbook of Topology.* Academic Press, 1980.

[24] A. Shesh and B. Chen. Peek-in-the-Pic: Flying Through Architectural Scenes From a Single Image*. *Computer Graphics Forum*, 27(8):2143–2153, 2008.

[25] M. Shpitalni and H. Lipson. Identification of Faces in a 2D line drawing projection of a wireframe object. *IEEE Trans. PAMI*, 18(10):1000–1012, 1996.

[26] K. Sugihara. Mathematical structures of line drawings of polyhedrons-toward man-machine communication by means of line drawings. *IEEE Trans. PAMI*, (5):458–469, 1982.

[27] K. Sugihara. A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Trans. PAMI*, (5):578–586, 1984.

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

[29] A. Vicent, P. Calleja, and R. Martin. Skewed mirror symmetry in the 3d reconstruction of polyhedral models. *Journal of Winter School on Computer Graphics*, 11(3):504–511, 2003.

[30] Y. Wang, Y. Chen, J. Liu, and X. Tang. 3D reconstruction of curved objects from single 2d line drawings. *CVPR*, pages 1834–1841, 2009.

[31] T. Xue, J. Liu, and X. Tang. Object cut: Complex 3D object reconstruction through line drawing separation. *CVPR*, 2010.