Algorithms for the recovery of 3D objects from 2D line drawings

Fang, Fen

2014

Fang, F. (2014). Algorithms for the recovery of 3D objects from 2D line drawings. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/61726>

<https://doi.org/10.32657/10356/61726>

Algorithms for the Recovery of 3D Objects from 2D Line Drawings

Submitted By: Fang Fen

Supervisor: A/P Lee Yong Tsui

School of Mechanical & Aerospace Engineering

Nanyang Technological University

Singapore

August 2014

Abstract

Reconstructing the 3D object depicted in a line drawing consistent with human perception is an interesting and important topic in computer vision. Normally, the reconstruction process requires two main steps: topological analysis and 3D recovery. This thesis makes three novel contributions in the three steps: (1) Decomposition of the line drawing of a complex object into a collection of drawings of simple manifolds; (2) Finding the faces of the object in the drawing; (3) Reconstruction of the object in both parallel and perspective projections. The work here is restricted to drawings of planar polyhedra.

The decomposition of a complex line drawing into drawings of simpler manifolds occurs at (1) non-manifold vertices, (2) along non-manifold edges and (3) across internal faces. The algorithms for the first two are simple. The decomposition across an internal face has two stages: basic decomposition and the subsequent repair. Our results show that the algorithm can deal with objects with planar faces. Reconstructing of the 3D object is much easier and much more efficient from a simpler drawing than a complex one.

Knowing the faces of the object in a drawing is essential to the 3D reconstruction of the object. A new face finding algorithm much more efficient than the existing ones is presented here. This algorithm also contains two stages: decomposition and face forming. In the first stage, a line drawing is decomposed into sub-loops, which are connected but usually open sequence of edges that must belong to the same face. Faces are formed from the sub-loops in the next stage. The method has the capability to differentiate an internal face from a real one once it is formed.

After obtaining the faces, the next step is to reconstruct the 3D model. Here, we present a novel, simple and direct method, the cubic corner method (CCM). A cubic corner is a trihedral vertex where the three incident edges are mutually perpendicular. The 3D coordinates of the three end vertices relative to the given vertex can be obtained analytically given the 2D coordinates of the four vertices in the drawing. The algorithm starts with a user-identified cubic corner in the drawing and uses the face information obtained previously to recover the geometry of the entire object. It works for both parallel and perspective drawings; the only difference is that, for perspective, the method needs to find the focal length first, to establish the linear relationship between a vertex on the image plane and its corresponding point on the object. The method generates accurate results for accurate drawings. For inaccurate drawings, it generates plausible results which may not be good enough. A novel hybrid method, combining CCM and the well-known optimization-based method (OBM), overcomes the problem. The optimization process in OBM requires an initial guess to the 3D object; a guess close to the final solution could improve the result of OBM significantly. In the hybrid method, CCM supplies this initial guess to OBM, which then produces a result better than either method on its own.

This thesis also presents an alternative efficient method that recovers 3D objects from perspective drawings by reversing the process one uses for drawing a perspective drawing manually. The method does not require the focal length and, depending on the topology of the drawing itself, may not need the faces of the object. This is a significant advantage as face identification is not a trivial task.

Future work is required to resolve some of the shortcomings in the algorithms presented in the thesis. Besides, there is a need to overcome the geometric restriction of planar polyhedra only, and extend the geometric coverage to objects that contain both planar and curved faces. Also, the ability to recover line drawings accurately opens the possibility of recovering objects in photographs, if we can extract edges accurately from photographs. This brings forth the need for a stitching algorithm that merges reconstructed results together to form the whole object as the complete view of an object normally requires multiple photographs. This can then lead to some useful applications such as the reconstruction of engineering objects accurately, without using the expensive 3D laser scanner. Much work remains before we can arrive at that.

Acknowledgement

There are many people I want to thank - everybody involved in my project, or my daily life in Singapore, has been helpful, positive, and supportive.

First and foremost, I would like to express deepest gratitude to my supervisor Associate Professor Lee Yong Tsui for his guidance, stimulating discussions and encouragement over the last three years. I am grateful to him for providing and sharing his innovative ideas in the fields of Computer Graphics.

I would like to thank my good friends, Ba Te, Yu Hao and Leong Mei Chee for their help in improving my English writing. I learned a lot from them. I also would like to thank my senior PhD student, Liu Yang, who gave me positive advice on my research.

I have made many friends, too many to name individually, among the research students of the school, especially those who share the responsibilities and fun of organizing activities for the MAE Graduate Student Club. They have given me many happy moments in my time at NTU.

The passion and friendship of the technical staff members of CAE Lab 1, Mr. Mike Leong, Ms. Low Sian Too, Ms. Ho Foon Siew and Mr. Chew Hock See make the lab an enjoyable place in which to work and study.

My family has been on my side all the time. They are always a source of energy and encouragement to me. I would like to take this opportunity to express my deepest gratitude to them.

Table of Contents

Abstract	I
Acknowledgement.....	IV
Table of Contents	V
List of Tables.....	XI
List of Figures	XI
List of Algorithms	XI
CHAPTER 1 Introduction.....	1
1.1 Background and Motivation	1
1.2 Objectives and Scope	3
1.3 Thesis Organization.....	3
1.4 General Assumption and Terminologies	4
CHAPTER 2 Literature Review.....	6
2.1 Parallel Projection	6
2.1.1 Line-Labeling based Method	6
2.1.2 Optimization-based Method.....	9
2.2 Perspective Projection	13
2.2.1 Geometry-based Method.....	13
2.2.2 Monocular Cues Method.....	15

2.3	Objects with both Curves and Straight lines	16
2.4	Chapter Summary	18
CHAPTER 3 Efficient Decomposition of Line Drawings of Connected Manifold		
	without Face Identification	19
3.1	Introduction	19
3.2	Related Work	19
3.3	Decomposition	22
3.3.1	Decomposition at Non-manifold Vertices	23
3.3.2	Decomposition along Non-manifold Edges	26
3.3.3	Decomposition across Internal Faces	27
3.3.4	A Special Case: High Degree Manifold Vertices Forming a Loop	42
3.4	Results	44
3.5	Chapter Summary	47
CHAPTER 4 A New Face Identification Method by Drawing Decomposition		
4.1	Introduction	49
4.2	Related Work	50
4.3	Terminologies	53
4.4	The Preprocessing and the Main Algorithm	54
4.5	Decomposition	57
4.5.1	Decomposing a line Drawing into Sub-loops	57

4.5.2	Properties of Sub-loops	65
4.6	Face Forming	68
4.6.1	Forming of Individual Faces	68
4.6.2	The Full Face Forming Algorithm	76
4.6.3	Redundant Sub-loops	77
4.6.4	Complexity Analysis	77
4.6.5	Internal Face Identification	79
4.6.6	Validation of Internal Faces	82
4.6.7	Correctness of the Results	83
4.6.8	Summary	83
4.7	Discussions	84
4.7.1	Non-manifolds	84
4.7.2	Drawings that Contain Multiple Internal Face Sets	87
4.7.3	Drawings that Contains Holes.....	88
4.7.4	Drawings with Multiple Interpretations	91
4.8	Results	91
4.8.1	Test Objects and Run Times	92
4.8.2	Comparison with Existing Method	93
4.9	Chapter Summary	98
CHAPTER 5 The Cubic Corner Method for 3D Reconstruction		100

5.1	Introduction	100
5.2	Cubic Corner	100
5.2.1	Cubic Corner in Parallel Projection	101
5.2.2	Cubic Corner in Perspective Projection	103
5.3	Recovery of 3D Objects	108
5.4	Discussion	113
5.4.1	Multiple Paths to a Vertex	113
5.4.2	Non- cubic Graphs	114
5.4.3	Non-cubic Corners	118
5.5	Results	122
5.6	Chapter Summary	127
CHAPTER 6 A Hybrid Method for 3D Object Recovery from 2D Drawings		130
6.1	Introduction	130
6.2	Analysis of the Cubic Corner Method and Optimization-based Method	130
6.2.1	Strengths and Weaknesses of the Cubic Corner Method	131
6.2.2	Analysis of the Optimization-based Method	133
6.2.3	Summary	134
6.3	A Hybrid Method for Inaccurate Drawing	135
6.3.1	The Importance of the Initial Guess for the Optimization	135
6.3.2	Testing Procedure	136

6.3.3	Results	137
6.3.4	Objects Which CCM Cannot Fully Recover.....	147
6.4	Discussion and Summary	149
6.4.1	Operational Domain	149
6.4.2	Comparison with Reported Works.....	150
6.4.3	Chapter Summary.....	152
CHAPTER 7 3D Reconstruction by Reverse Engineering of Drawings in 2-Point		
	Perspective View	155
7.1	Introduction	155
7.2	Outline of the Algorithm	155
7.3	Framework Construction	156
7.4	Obtaining the Plan View	159
7.4.1	Vertex Projection onto the Bottom Plane	160
7.4.2	Drawing the Plan View	164
7.5	Obtaining the Height of Each Vertex	168
7.6	Recovering the 3D Object	172
7.7	Results	173
7.8	Discussion	176
7.9	Chapter Summary.....	177
CHAPTER 8 Conclusion		179

8.1	Summary and Contributions	179
8.2	Conclusion and Future Work	182
	List of Publications	185
	References.....	186

List of Tables

Table 3.1	Time comparison between decomposing plus face finding and direct face finding for drawings in Fig. 3.17..	46
Table 4.1	Run time comparison between DBFS and DFF, time in milliseconds.	97
Table 5.1	The length ratio of the recovered object edges compared with the true values.	126
Table 7.1	Object 1 and object 2 are the black and orange box respectively in Fig. 7.8a.	175

List of Figures

Fig. 2.1	Line-labeled drawings.	7
Fig. 2.2	Inflation from 2D drawing using line-labeling based method.	8
Fig. 2.3	Reconstruction results from the line drawings	11
Fig. 2.4	Line drawings and their separation and reconstructed results.	12
Fig. 2.5	Complete 3D reconstruction of a real scene	15
Fig. 2.6	A set of line drawings, their transformed line drawings, and their reconstructed results, each shown from two viewpoints.	17
Fig. 3.1	Decomposition of a line drawing	21
Fig. 3.2	Drawings with non-manifold vertex, edge or internal face.	22
Fig. 3.3	A non-manifold vertex marked with a black dot and its adjacent vertices marked with circles.	23
Fig. 3.4	Two line drawings with non-manifold vertices and their decomposed simpler manifolds.	25
Fig. 3.5	A non-manifold edge and its adjacent vertices.	26
Fig. 3.6	Two line drawings with non-manifold edges and their decomposed simple manifolds.	27

Fig. 3.7	Two situations that result in incomplete parts.....	31
Fig. 3.8	A line drawing of a complex manifold.	33
Fig. 3.9	Two different decompositions of the line drawing in Fig. 3.8.....	34
Fig. 3.10	Examples illustrating isolated edges and vertices.	36
Fig. 3.11	Two line drawings and their decomposed simple manifolds.	37
Fig. 3.12	An example to demonstrate Step P1.	38
Fig. 3.13	An example.	40
Fig. 3.14	An example that cannot be decomposed.	41
Fig. 3.15	The partition of an object with degree-6 vertices forming internal faces.	43
Fig. 3.16	A set of examples and their decomposition with the computational times in milliseconds.	45
Fig. 3.17	Seven line drawings of increasing size denoted by seven loops.	46
Fig. 3.18	An object and its three options for decomposition.....	47
Fig. 4.1	Topologically non-manifold drawings that may be manifold geometrically....	54
Fig. 4.2	The decomposition of a block.	56
Fig. 4.3	A drawing with all its vertices of degree 4.	60
Fig. 4.4	A drawing with a vertex with degree of 4.....	60
Fig. 4.5	An example of decomposing a drawing.....	63
Fig. 4.6	A drawing which remains a manifold after some decomposing.	64
Fig. 4.7	Decomposition of a block	68
Fig. 4.8	A simple example.....	71
Fig. 4.9	Another example for illustrating the face forming.....	73
Fig. 4.10	An example which has multiple solutions.	74
Fig. 4.11	One line drawing with two different valid interpretations.	75

Fig. 4.12	Examples to show the appearance of internal faces.	80
Fig. 4.13	Internal faces divide sub-loops into several groups.....	81
Fig. 4.14	An example to illustrate internal faces identification.	82
Fig. 4.15	An object with a loop that has self-intersection.	85
Fig. 4.16	A non-manifold containing an edge shared by more than two faces.....	85
Fig. 4.17	Real faces that were regarded as internal faces in the face forming stage.	86
Fig. 4.18	A non-manifold which is treated as a manifold.....	86
Fig. 4.19	A drawing and its three possible internal face sets.....	88
Fig. 4.20	The two loops at the bottom are not treated as in the same plane.	89
Fig. 4.21	Two possible decomposition results for a drawing.	90
Fig. 4.22	Objects with multiple valid interpretations.	91
Fig. 4.23	Test objects without internal faces..	93
Fig. 4.24	Test objects with internal faces.	94
Fig. 4.25	Buildings, desks and stairs.....	94
Fig. 4.26	The relation between time and the number of edges in buildings, desks and stairs.....	94
Fig. 4.27	Examples used to compare the efficiency between DFF and DBFS.....	98
Fig. 5.1	A cubic corner and its 2D projection.....	101
Fig. 5.2	A cubic corner and its projection on the image plane.	103
Fig. 5.3	Geometric construction of the focal length with the triangle of vanishing points.	105
Fig. 5.4	An object with seven faces. The chosen cubic corner is marked with a dot. .	109
Fig. 5.5	Faces adjacent to a given face.	112
Fig. 5.6	Inaccurate drawings and the recovered 3D objects.	114

Fig. 5.7	Objects with non-cubic graph.	116
Fig. 5.8	Face adjacency graphs for the objects in Fig. 5.7c and Fig. 5.7d.	117
Fig. 5.9	Non-manifold objects.	118
Fig. 5.10	Parallel projection of a non-cubic corner.	119
Fig. 5.11	Objects recovered from accurate line drawings.	125
Fig. 5.12	Objects recovered from (a) two-point and (b) three-point perspective drawings.	126
Fig. 5.13	3D objects recovered from photographs.	126
Fig. 5.14	Nearly identical projections of three blocks of different lengths.	128
Fig. 5.15	Ambiguity: which are the cubic corners, a & b or c & d ?	129
Fig. 6.1	Results from inaccurate drawings using CCM.	131
Fig. 6.2	CCM with inaccurate inputs and resulting 3D models.	132
Fig. 6.3	Two figures showing results from OBM	133
Fig. 6.4	Flow chart of the hybrid method.	135
Fig. 6.5	Four line drawings at four different accuracies and the recovered objects using CCM, OBM and the hybrid method.	140
Fig. 6.6	The recovery of another object.	141
Fig. 6.7	Another example.	143
Fig. 6.8	An example from Lipson and Shpitalni	144
Fig. 6.9	The same inputs as in Fig. 6.8.	145
Fig. 6.10	An example due to Varley and Company	146
Fig. 6.11	Another example due to Varley and Company	147
Fig. 6.12	A non-simple manifold which CCM cannot fully recover.	148
Fig. 6.13	Objects with inaccessible vertices in CCM.	149
Fig. 7.1	Two-point perspective view in 3D and in spread out plan view.	159

Fig. 7.2	Illustration of projection to the bottom plane.	161
Fig. 7.3	Projection onto the bottom plane by using the three rules and face information.	163
Fig. 7.4	An illustration of the process of drawing the plan view.	164
Fig. 7.5	Left plane projection.	170
Fig. 7.6	Illustration of obtaining the height of vertices.	172
Fig. 7.7	Objects recovered from accurate perspective drawing.	174
Fig. 7.8	3D models recovered from single image with lines drawn on the edges of desired objects.	175

List of Algorithms

Algorihtm 3.1	Decomposition at a non-manifold vertex.	24
Algorihtm 3.2	Forming of a simple manifold.	29
Algorihtm 3.3	Decompositon across internal face.	36
Algorihtm 3.4	The whole algorihtm	42
Algorithm 4.1	The decomopsition algorithm.	64
Algorihtm 4.2	Face forming algorithm.	77
Algorihtm 4.3	The face forming algorithm for general objects.	88
Algorihtm 5.1	Reconstruction of polyhedral objects from an accurate line drawing..	112
Algorihtm 7.1	3D object reconstruction from two-point perspective line drawing.	172

CHAPTER 1

Introduction

1.1 Background and Motivation

There are many powerful CAD systems like Pro/Engineer, Solidworks, Autodesk Inventor and so on. A new version of each system is released periodically, more sophisticated and more user-friendly. Even then, during conceptualization, designers still would rather use pencil and paper than CAD systems because the need to select icons and menu options, to specify dimensions and other data interferes with creativity and the flow of thought. However, in practice, designing using only paper and pencil is inadequate because there is completely no interaction between the designers and their work, and a drawing on paper has little use beyond being looked at. For example, a drawing cannot be translated, scaled and rotated on paper. Only one view of the object can be depicted in a drawing; multiple views require multiple drawings. Basically, current CAD systems allow users to interact with drawings conveniently, but they are not suitable in the early stage of a design when the ideas are still fluid. Sketching on paper is common in the early stage, but it does not allow users to interact with the drawings.

It would be good for designers to have a system possessing conveniences of both paper-based sketching and CAD systems, allowing creation and regeneration of ideas dynamically while capturing the details of the design. After sketching an object, the user can take it into 3D for further operations, such as scaling, rotation, chamfering and filleting; one can also go back to the 2D drawing, now enhanced, to further the design.

Many researchers have been working on this subject over the last three decades and a number of sketch-based systems have been developed. One leading work is that of Lipson and Shpitalni [1], which recovers 3D objects from 2D line drawings containing straight and/or curved lines by using an optimization-based method. However, its computational efficiency is poor and convergence to the correct solution cannot be guaranteed for a complex drawing, which is a deficiency inherent in the optimization process. Liu et al. [2] improved the optimization-based method by changing the variables of the objective function for the optimization process. Their system works better but still cannot handle complex line drawings well. Recently, Chen et al. [3] proposed a method to recover 3D objects by performing three steps: firstly, decompose a complex line drawing into simpler ones across internal faces; secondly, recover these decomposed drawings separately; finally, merge all the recovered objects to form the one represented by the original drawing. This system can handle objects which are complex, but it requires finding internal faces, which means finding all the faces of the object in the drawing, which can be an expensive process. The same problem is also present in their subsequent work [4]. Therefore, their system can save time for the 3D reconstruction but requires time for face finding.

To develop a 3D reconstruction system, the computational efficiency of both processes, face finding and 3D reconstruction, deserve more attention, especially for complex objects. Systems reported in [3] and [4] save time in the 3D reconstruction, but take time in the face finding. In addition, all optimization-based 3D recovery methods cannot guarantee the accuracy of the resulting 3D objects. It is necessary to develop an efficient system for face finding as well as 3D recovery, and improve the quality of the outcome.

1.2 Objectives and Scope

The ultimate aim of the research work reported in this thesis is a system for producing 3D models from 2D line drawings, which can be incorporated into CAD systems, as a new natural user interface for conceptual design. Designers are to be able to communicate freely with their computer using such an interface.

A complete 3D recovery system should deal with four key issues: input accuracy, mode of projection, with or without hidden lines and the number of views required.

An input drawing can be accurate (such as those exported from a CAD system) or inaccurate (such as a sketch). It can be drawn in either parallel or perspective projection. Hidden lines may or may not be shown. Definitely, reconstruction from inputs without hidden lines is more difficult as some information would be missing. The full information on an object may be conveyed in one or multiple views.

The work in this thesis leads part way to that objective; it handles both accurate and inaccurate inputs, and deals with drawings in both parallel and perspective projections. However, it only studies inputs with hidden lines in one single view of an object.

There are three major tasks in this thesis: decomposing a line drawing into simpler ones based on topological analysis, face finding and 3D recovery.

1.3 Thesis Organization

This thesis contains eight chapters including Chapter 1: “Introduction”.

Chapter 2 gives a review of previous studies on topological analysis, face finding and 3D reconstruction methods from 2D line drawing.

Chapter 3 introduces a novel method to decompose a complex line drawing into simpler ones without finding faces.

Chapter 4 presents a new method to find faces by decomposing an input drawing into chains of edges, with each chain belonging to one face, followed by forming faces using these chains.

Chapter 5 proposes a direct method to recover the geometry of the 3D polyhedron with at least one cubic corner depicted in a single parallel or perspective projection.

Chapter 6 describes a hybrid method to recover 3D polyhedral objects from 2D line drawings by combining two known methods, the cubic corner method and the optimization-based method.

Chapter 7 discusses a different algorithm to recover the 3D object depicted in an accurate 2D line drawing in a 2-point perspective view. It reverses the perspective drawing process which creates a perspective view from two orthogonal planar projections.

Chapter 8 provides an overall summary of the thesis, draws some conclusions and discusses the future work.

1.4 General Assumption and Terminologies

In this thesis, all inputs are assumed to be line drawings in a generic view, in which all the edges and vertices are visible, no edges overlap and no edge is projected into a point.

In this thesis, two words, “reconstruction” and “recovery” are used interchangeably.

Nine terms that appear in multiple chapters in this thesis are introduced here.

Chord: A chord is an edge that connects two non-adjacent vertices in a loop.

Virtual chord: A virtual chord of a cycle is an imaginary straight line connecting two non-adjacent vertices of the cycle. It does not appear as an edge in the drawing.

Manifold: A manifold is a solid where every point in its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space [5].

Complex manifold: A complex manifold is a manifold that can be decomposed into simpler manifolds at existing vertices, along existing edges or across internal faces.

Simple manifold: A simple manifold is a manifold that cannot be decomposed into simpler manifolds at existing vertices, along existing edges or across internal faces.

Simple non-manifold: A combination of sheet faces to form a non-solid object. Simple non-manifold is a subset of non-manifold [6].

Degree of vertex: The degree of a vertex is the number of edges meeting at the vertex.

Rank of edge: The rank of an edge is the number of faces sharing the edge.

Internal face: An internal face could be thought of as the face formed by gluing two faces of two manifolds together to form one manifold, and is thus internal to the combined manifold and is not a true face.

CHAPTER 2

Literature Review

3D reconstruction from 2D line drawing has been studied for decades, and much work has been done. As indispensable components of 3D reconstruction, topological analysis and face finding of 2D line drawings also caught researchers' attention. In this chapter, only previous work related to 3D reconstruction from 2D drawings will be presented. They are organized based on two projection modes: parallel and perspective; input drawing includes straight lines and curves. Related work of the components will be reviewed later in the specific chapters.

2.1 Parallel Projection

Under parallel projection, projection rays, which are lines projecting points on the objects to the projection plane, are parallel. A parallel projection is simply obtained by dropping the z component of the 3D object. Two main methods, based on line-labeling and optimization, have been applied in 3D reconstruction from drawings in parallel projection.

2.1.1 Line-Labeling based Method

Line-labeling is a well-established preliminary stage of interpreting line drawings [7], [8], [9]. As shown in Fig. 2.1, each line in a drawing can be labeled as convex (marked by + in Fig. 2.1), concave (marked by -) or occluding (marked by $\rightarrow$). Each line L in a 2D drawing actually is the projection of a 3D edge, which is the intersection of two planar faces S_1, S_2 . In the case of only one of S_1, S_2 is visible in the drawing, line L is labeled as

occluding. Otherwise, L is concave or convex depending on whether the exterior angle between S_1 and S_2 is less or greater than π respectively [10].

Fig. 2.1. Line-labeled drawings [11].

The line-labeling technique in itself does not recover the 3D shape, it was originally proposed to identify and reject impossible drawings [7], [8]. But the technique is able to create facilities for interpreting line drawings. First, the line-labeling technique can be used to differentiate edges, so that we know which edges bound the visible faces or partial faces of the object and which merely occlude them. Second, it can be used to constrain the possibilities when trying to recover the topology of the hidden part of the object based on the underlying vertex types implied by the junction labels. Third, it can be used to constrain the geometry of an edge which is to be extended or added by labeling on junctions. With these advantages, the line-labeling could be a useful input to the process of reconstruction [12].

Line junction can be categorized into four types, L, Y, W, T, according to the geometry of the intersection at the junction. Lines meeting at a junction can be labeled in a finite number of ways. These finite outcomes, known as the Huffman-Clowes catalogue, were initially derived independently by Huffman [7] and Clowes [8]. Each line segment in a drawing can be labeled in accordance with the line-labeling rules. Using a library of

possible junction configurations, a set of consistent line-labels is determined and then reconstructed into 3D using additional information. Kanade [13] and Sugihara [14] perform 3D reconstruction using line-labeling with image regularities, which are geometric conditions that can be extracted from 2D drawings and are preserved in 3D; examples include line parallelism and skewed symmetry.

Cooper [15] uses a novel line-labeling technique to reconstruct wireframes from a single view. Cooper's system is able to handle straight as well as curved lines. Martin et al. [11] proposed to perform 3D reconstruction from a single view by combining the line-labeling scheme with Boundary-representation (B-rep). Fig. 2.2 shows an example of a 3D wireframe reconstructed using a method based on line-labeling.

Fig. 2.2. Inflation from 2D drawing using line-labeling based method [11].

Line-labeling is useful in the interpretation of line drawings; however, it is not directly a 3D reconstruction method. It usually needs to combine with additional information to perform 3D reconstruction and it is only applicable to visible lines in a drawing. This means that line-labeling based reconstruction methods are generally suitable for polyhedra without hidden lines. In addition, the methods are not able to handle inaccurate drawings and possible missing entities.

2.1.2 Optimization-based Method

In mathematics, optimization means solving problems in which one seeks to minimize or maximize an objective function by systematically choosing the values of the variables from an allowed set. In 3D reconstruction, optimization means seeking the unknown z-coordinates of the vertices to minimize the objective function $F(z)$ under some soft constraints, which are image regularities that contain the z coordinates of the relevant vertices as variables. $F(z)$ takes the form

$$F(z) = [w][\alpha]$$

where $[\alpha] = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$ is a vector containing the contributions of individual regularities. And $[w] = [w_1, w_2, \dots, w_n]$ is a vector of the weights which are attached to the respective regularities based on the strength of their effect on the reconstruction, and n is the number of regularities used.

The optimization-based method was first used by Marill [16]. In his work, the z-coordinates of the vertices of a 3D object represented by a line drawing is repeatedly computed until the minimum standard deviation of angles (MSDA) between line pairs at junctions is reached. But Marill's system fails when the input drawing contains two or more significantly different angles. Motivated by Marill's system, Brown and Wang [17] proposed to minimize the standard deviation of the segment magnitudes (MSDSM), and Shoji et al. [18] presented the criterion of minimizing the entropy of angle distribution (MEAD). Leclerc and Fisler [19] improved the optimization-based method by introducing one more image regularity, face planarity, to the objective function. The performance of their system is better than that of using only MSDA, MSDSM or MEAD

as the constraint in the objective function. However, two image regularities, MSDA and face planarity, seem only sufficient for reconstructing a limited set of object types.

Lipson and Shpitalni [1] extended Leclerc and Fischler's method by adding more image regularities, such as line verticality, line parallelism, isometry, corner orthogonality and skewed facial orthogonality. Inspired by Lipson and Shpitalni, Piquer et al. [20] used a symmetry constraint to recover objects with mirror symmetry. Lipson and Shpitalni's method can handle more complex objects than previous methods. But its computational efficiency is poor because it involves too many constraints, resulting in slow convergence and possibly arriving at a local minimum which means producing an incorrect result. To remedy the inadequacies of Lipson and Shpitalni's method, Liu et al. [2] proposed a new reconstruction system by replacing the z coordinates of vertices with parameters of the planes that contain the planar faces of an object as the variables of the objective function. This leads to a set of linear constraints on the planes of the object, resulting in a much lower search space where optimization is easier to achieve. This method can reconstruct more complex 3D objects from 2D line drawings as well as is computationally more efficient compared with Lipson and Shpitalni's method. Fig. 2.3 shows reconstructed 3D objects from LFA (Leclerc and Fischler's algorithm), LSA (Lipson and Shpitalni's algorithm) and POA (Plane-based Optimization algorithm).

Wang et al. [21] used the optimization-based method to reconstruct 3D objects from 2D line drawings with curved line segments. Three more image regularities, curve parallelism, generalized face perpendicularity and curve concavity¹, are used. Chu et al. [22] proposed a novel optimization approach by replacing the z coordinates of the

¹ Concavity of curved edges means the edges lie on the same curve, akin to collinearity of straight edges.

vertices with the parameters of lines as the variables of the objective function. This approach is also suitable for recovering 3D objects from perspective line drawings.

Fig. 2.3. Reconstruction results from the line drawings a-c: a1-c1 by LFA, a2-c2 by LSA, and a3-c3 by POA [2].

Chen et al. [3] proposed a divide-and-conquer method to recover 3D objects from line drawings. Their method first decomposes a complex line drawing into multiple simpler ones across internal faces. The decomposed line drawings are then recovered separately

using the optimization-based method, and the results are merged together to form the complete object depicted in the original line drawing. The method can handle complex objects more efficiently than all preceding methods and reduce the computation time for 3D reconstruction significantly. However, the method requires finding internal faces first, which is not trivial before decomposition. Xue et al. [4] proposed that a complex line drawing can be partitioned into simpler ones across “cuts”. Those partitioned drawings are reconstructed utilizing an optimization-based method. Like the method in [3], the method in [4] also requires face finding first. Fig. 2.4 shows two line drawings, their separation and reconstruction results using the divide-and-conquer method.

Fig. 2.4. line drawings and their separation and reconstructed results [3].

On the whole, the optimization-based method works well. The 3D objects reconstructed using this method are usually consistent with human perception. Its biggest disadvantage results from an innate property of the optimization process, which is that the searching space varies like n^3 , where n is the number of vertices in the object [1]. Therefore, the convergence is slow and the ability to converge to the global minimum is poor when the 2D line drawing is complicated. Some progress has been made in these respects. For

instance, Lipson and Shpitalni [1] achieved improved performance by adopting Angular Distribution Graph (ADG) analysis. ADG is constructed by sampling the angle of each entity in the sketch and plotting it on an angular distribution histogram, to generate better initial input to the optimization process. Liu et al. [2] and Chu et al. [22] reduced the search space by replacing the z coordinates of the vertices with plane parameters and line parameters as the variables of the objective function respectively. They did not solve the root problem, because they did not deal with one key factor which greatly affects the optimization results, and that is the initial guesses for the variables. The difference between the initial guess and the target has a great bearing both on the computational time and the ability to converge to the correct result. The initial guess is a preliminary estimation of the shape of the 3D object depicted by the 2D line drawing. A better initial guess for the optimization-based method should improve the 3D reconstruction, in terms of speed and quality. This is one area we will be addressing in this thesis.

2.2 Perspective Projection

In perspective projection, parallel lines which are not parallel to the projection plane converge at one point, the vanishing point. Perspective projections are classified into one, two and three-point perspective according to the number of vanishing points in the three principal x , y and z directions. There are two main methods for reconstructing 3D objects from perspective projections: geometry-based method and monocular cues method.

2.2.1 Geometry-based Method

The Geometry-based method utilizes the properties of the perspective world in the picture, such as parallel lines, vanishing points and camera focal length and so on. This method

usually is used to recover 3D objects from perspective projections in wireframe or images with lines drawn on the edges of the desired objects.

There has been some work in recovering 3D objects in perspective projection by first transforming perspective projection to parallel projection [23], [24], [25], [26]. To perform this procedure, the key point is to estimate the focal length. Therefore, some basic geometry information of perspective drawing is required, for example, vanishing points and vanishing lines. After obtaining the focal length, the x , y coordinates of a vertex in the object can be expressed with the z coordinate as the only unknown, then, the optimization process is employed to get the optimal z coordinates with the help of image regularities, such as face planarity, line parallelism, diagonal isometry and corner orthogonality [23], connectivity, perspective symmetry and common plane [26].

Criminisi et al. [27], [28], [29] proposed a measurement method based on the basic geometry of perspective drawing, the measurement method is independent of the camera's internal parameters - focal length, aspect ratio, principal point and skew. It is assumed that the vanishing line of a reference plane in the scene can be determined from the image and a vanishing point for another reference direction. Under such conditions, the method is able to handle three types of measurement: measurement of the distance between any of the planes which are parallel to the reference plane; measurements on these planes; and determining the camera's position in terms of the reference plane and direction. Therefore, once the reference plane is chosen, its distance to each plane of the object can be determined using the measurement method; hence, the 3D scene can be reconstructed. Fig. 2.5 shows a reconstructed scene using this method.

Fig. 2.5. Complete 3D reconstruction of a real scene: (a) original image; (b) a view of the reconstructed 3D model; (c) another view of the reconstructed 3D model [28].

On geometry-based methods, we introduced two algorithms. The first one depends on the camera internal parameter, focal length, and uses the optimization process to get the optimal z coordinates. Therefore, this method cannot produce the exact solution even if the input drawing is accurate. The second is the Criminisi's algorithm. It is independent of the camera's internal parameters, but it requires user interactions to indicate the reference plane and direction.

2.2.2 Monocular Cues Method

Monocular cues include convergence of parallel lines, foreshortening, occlusion, shadows, aerial haze, texture variation, texture gradient and knowledge of the object geometry, all of which contribute to depth prediction [30]. For instance, texture on objects look different at different distances from the viewer. Texture gradients can help to indicate depth as well by capturing the distribution of the directions of edges.

Much has been done in reconstructing 3D objects using monocular cues. Horn and Brooks [31] developed the Shape from Shading (SFS) technique to recover 3D objects. Based on Horn and Brooks's work, Zhang et al. [32] showed that shading combined with some other cues can improve the results of existing SFS algorithms. Blake et al. [33] and

Loh et al. [34] proposed shape from texture methods to reconstruct objects with curved surfaces from images.

In recent years, Saxena et al. [35], [36] used a novel model to reconstruct objects from a single image. The model uses a discriminatively-trained Markov Random Field (MRF) that includes multi-scale local and global-image features, and models both depths at individual points as well as the relationship between depths at different points.

A 3D scene can be reconstructed from a single image directly with the help of monocular cues. However, the monocular cues heavily rely on prior knowledge learned from training sets about the environment. Thus, the monocular cues may not generalize well to images quite different from the ones in the training set.

2.3 Objects with both Curves and Straight Lines

To handle objects which contain both planar and curved faces, Masry et al. [37] presented a progressive way to recover 3D object from 2D drawings with simple curved edges. The method has a limitation that the free hand sketches must not have strokes with large deviations from the underlying axis system. The system of Cao et al. [38] also can do 3D reconstruction of objects with both planar and curved faces, but it requires users to add lines on the curved faces. Wang et al. [21] added three image regularities based on [1] to handle shapes with curved faces. In their work, a curved edge was converted to two straight lines by connecting its start point, singular point and end point successively. Then, they employ the optimization-based method to recover the converted wireframe with straight lines only. The complex object is treated using the algorithm described in [3]. It works well for some complex objects which can be divided into multiple simpler

shapes; but for those complex objects which cannot be divided, the results are unsatisfactory. Fig 2.6 shows some line drawings, their transformed line drawings, and their reconstructed results using the algorithm in [39].

Fig. 2.6. A set of line drawings, their transformed line drawings, and their reconstructed results, each shown from two viewpoints [39].

In the work of Liu and Lee [40], an object with curved faces was transformed into a polygon by converting a curved face to series of planar faces using Dynamic Timing Warping (DTW), the optimization-based method was used to do 3D reconstruction. This method will increase the number of faces; it is not suitable for objects with many curved faces. Oswald et al. [41] came up with a novel algorithmic solution for estimating a three-dimensional model of an object observed in a single image. Their algorithm interactively determines the objects' silhouette and subsequently computes a silhouette-consistent 3D model based on a minimal user input. Comparing with other approaches, the proposed algorithm does not constrain the resolution in the depth-direction. Shao et al. [42] presented an approach to facilitate the creation of 3D-looking shaded production drawings from concept sketches. The key to their approach is a class of commonly used construction curves known as cross-sections that help in both sketch creation and viewer

understanding of the depicted 3D shape. Experimental results show that their approach enables users to shade a wide range of concept sketches with a variety of rendering styles.

2.4 Chapter Summary

The complete process of 3D reconstruction from 2D line drawings includes: input drawing tidying, decomposition, face finding and 3D reconstruction. Drawing tidying is a trivial job; it will not be studied in this thesis. Although much work has been done in the area of decomposition and face finding, the latest methods in the area are not efficient enough. There is plenty of room for improvement. Related works and a new method of decomposition will be introduced in Chapter 3, those of face finding will be presented in Chapter 4.

In this chapter, relevant work on 3D reconstruction is reviewed. These contributions are useful and significant to 3D reconstruction. There are also gaps in these methods. For example the line-labeling based methods [7-11, 14] are not able to handle polyhedral with hidden lines and is not suitable for handling inaccurate drawings; the optimization-based methods [16, 17] suffer from the innate properties of the optimization process, which are poor computational efficiency, uncertainty of convergence to the correct solution and always providing approximate rather than accurate solutions.

Getting rid of these gaps is our ultimate goal and forms the significant contributions of this thesis. Two new 3D reconstruction methods will be introduced in Chapters 5 and 6.

CHAPTER 3

Efficient Decomposition of Line Drawings of Connected Manifold without Face Identification

3.1 Introduction

This chapter presents an algorithm for decomposing complex line drawings which depict connected 3D manifolds into multiple simpler drawings of individual manifolds. The decomposition process has three stages: decomposition at non-manifold vertices, along non-manifold edges and internal faces. Once non-manifold vertices and/or edges are found, the decomposition can be performed straightforwardly. Thus the major task in this chapter is decomposition across internal faces. This has two steps: basic decomposition and repair of incomplete parts. The decomposition is performed before face identification which is computationally expensive. After decomposition, the time for face finding is much reduced and this, in turn, greatly improves the process for 3D reconstruction, which is the ultimate goal.

3.2 Related Work

There are four works in the literature on the partition of line drawings. Sun and Lee [43] analyzed the topology of a line drawing that may contain manifold and non-manifold entities. Manifolds may be joined at vertices or edges to form non-manifolds. Sun and Lee provided algorithms to determine the existence of such non-manifolds without actually partitioning them into simpler manifolds. Manifolds may be joined at faces, either fully or partly, resulting in internal faces. They also defined an internal face as a face with all its edges ranked higher than 2, and proposed that a line drawing may be

decomposed along an internal face. But their definition is not complete; for example, the internal face f_1 in Fig. 3.1a will not be identified by their definition.

Chen et al. [3] proposed a divide-and-conquer method to recover 3D objects from line drawings. This method first decomposes a complex line drawing into multiple simple manifolds by using internal faces, which are identified in the face finding process. The decomposition of a manifold across an internal face produces two simpler manifolds. The decomposed line drawings are then recovered separately and the results are merged together to form the complete object depicted in the original line drawings. This method can handle complex objects more efficiently than other known methods. Even though the computation time for 3D reconstruction is much reduced in [3], face finding is still required, with its ensuing high computational cost.

Liu et al. proposed several properties that are used to identify an internal face in a line drawing [44]. The line drawing can be decomposed easily across the internal faces once they are identified.

Xue et al. [4] proposed that complex line drawing can be decomposed across “cuts”. A cut is a planar cycle consisting of some edges of the line drawing and/or new edges. An internal face is a special case of a cut. These two decomposition methods require knowing the faces beforehand. Therefore, the partitioning can reduce the time for 3D reconstruction but not for face finding, since the decomposition requires the faces to be found first.

This chapter proposes a novel method for decomposing complex line drawings without requiring face information. It is important to note that vertices exist explicitly when edges

in the same plane intersect and the decomposition is made at existing vertices and edges, without introducing any new ones.

Fig. 3.1. Decomposition of a line drawing. (a) A line drawing of a non-manifold. The black dot is a non-manifold vertex; the bold black line is a non-manifold edge; the shaded polygons, f_1 and f_2 , are internal faces. (b) Five simple manifolds are obtained by decomposing at the non-manifold vertex, non-manifold edge and internal faces.

We introduce here the special terms that appear in the rest of the chapter. The diagram in Fig. 3.1 shows the decomposition of a line drawing and illustrates some of the terms.

Non-manifold vertex: A non-manifold vertex is a vertex where a non-manifold can be decomposed to produce multiple simpler manifolds.

Non-manifold edge: A non-manifold edge is an edge along which a non-manifold can be decomposed to produce multiple simpler manifolds.

Isolated vertex: A vertex that belongs to only one simple manifold after the decomposition of a complex manifold.

Complete part: A complete part is a decomposed part that contains vertices with degree of 3 or more.

Incomplete part: An incomplete part is a decomposed part that contains vertices with degree 2 or less.

The decomposition of a complex manifold or non-manifold must result in a collection of simpler manifolds and, in the best case, a collection of simple manifolds. There should not be any dangling edges containing vertices with degree 1 or faces containing vertices with degree 2. This is a condition that we use to check for the correctness of the decomposition, because a drawing with all its vertices having degree 3 or more can be considered as a manifold topologically [6], [43].

3.3 Decomposition

There may be non-manifolds connected at a vertex or an edge or a manifold connected at a face, as shown in Fig. 3.2. When the connection is at a vertex, the vertex is a non-manifold vertex and has a degree of at least six [43]. When connection is at an edge, the edge is a non-manifold edge, and each of its end vertices normally has a degree of at least 5 [43]. In the special situation when the non-manifold edge is collinear with another incident edge, an end vertex may have degree 4. When the connection is at a face, the face is called as internal face [44].

Fig. 3.2. Drawings with non-manifold vertex, edge or internal face.

The decomposition aims to produce a collection of simple manifolds, and the process has three stages according to the type of connection: decomposition at non-manifold vertices, non-manifold edges and internal faces. These stages are carried out sequentially. The

decomposition here is along existing vertices and edges; it does not create or delete either. Therefore, the vertices and edges in each separated component are inherited from the original line drawing.

The input consists of a vertex adjacency matrix indicating which two vertices have an edge connecting them, and an edge connectivity list showing the list of edges connect to a given vertex. The latter can be derived from the former but is stored explicitly for computational efficiency.

3.3.1 Decomposition at Non-manifold Vertices

Any vertex of degree six or greater is a potential non-manifold vertex (NMV) [43]. In our method, decomposing a line drawing at a NMV requires finding all the adjacent vertices of the NMV, which can be inferred from the edge connectivity list. For example, Vertex 0 in Fig. 3.3 is a NMV, with six adjacent vertices, 1, 3, 7, 8, 12 and 14. A drawing may have multiple NMVs; all of them are detected and stored in a set which we call U .

Fig. 3.3. A non-manifold vertex marked with a black dot and its adjacent vertices marked with circles.

Vertices belonging to one manifold must have a connecting path that does not pass through any vertex in U . Hence, in Fig. 3.3, Vertices 11 and 14 belong to the same manifold because there exists a path, such as 11-8-9-14, connecting them without

containing Vertex 0, the NMV. Conversely, Vertices 3 and 14 belong to two different manifolds because every path that connects them contains Vertex 0. Hence, to form a manifold, the algorithm simply goes round collecting vertices that are connected without going through any vertex in U . It does so until all the vertices have been collected into individual manifolds. The process is given in Algorithm 3.1, in which a vertex is said to be open if the edges connected to it have not been traversed. It requires the NMV and the list of vertices directly adjacent to the NMV as input.

Algorithm 3.1: Decomposition at a non-manifold vertex.

- 1) *Let s be an empty set.*
- 2) *Put one adjacent vertex of the NMV into s .*
- 3) *For every open vertex in s*
- 4) *For every edge connected to this vertex*
- 5) *If the other endpoint of the edge is not the NMV and does not exist in s ,
Store the endpoint in s*
- 6) *End For*
- 7) *End For*
- 8) *Add to s the NMV and other vertices in U with one or more adjacent vertices already in s*
- 9) *Store the vertices in s as a separate manifold*
- 10) *While the number of unused adjacent vertices of the NMV is not 0,*
- 11) *Move to the next unused adjacent vertex*
- 12) *If this vertex does not belong to an existing manifold*
- 13) *Empty s*
- 14) *Put this vertex in s and go to step 3*
- 15) *End If*
- 16) *End While*

This algorithm will visit every potential NMV, i.e. a vertex with degree six or greater and return separate manifolds with all their vertices. If the NMV is a manifold vertex, then a single component is returned, except for a special case dealt with in Section 3.3.4.

The next step is to reorganize the structure and contents of the vertex adjacency matrix and the edge connectivity list. For the object in Fig. 3.3, instead of one matrix and one list, there are now two each, for the two simpler manifolds. Their contents are simply inherited from the values in the original. This reorganization process also needs to take place after decomposing a part along non-manifold edge and across an internal face.

Since Algorithm 3.1 visits every vertex only once, its complexity is $O(n)$, where n is the number of vertices. Upon exiting from Algorithm 3.1, a component may contain other NMVs. It is then necessary to apply Algorithm 3.1 again to each component, until every NMV is processed. Hence, the overall algorithm complexity is the sum of the complexities in applying Algorithm 1, each time with a smaller n . In the worst case, all the vertices are potentially NMVs, giving a complexity of $O(n^2)$. On average, we can expect n to be halved each time, and then the over complexity would be $O(n+n/2+n/4+\dots)$ which tends to $O(n\log n)$. Fig. 3.4 shows two examples of decomposition at non-manifold vertices.

Fig. 3.4. Two line drawings with non-manifold vertices and their decomposed simpler manifolds.

This algorithm differs from that of Sun and Lee [43] where the task was merely to identify the existence of a non-manifold vertex. Their work was done by checking whether all the vertices immediately adjacent to the NMV can be joined by one single

loop of edges, in which case, it is a manifold vertex. Their algorithm cannot be used for decomposition because it does not reach out to the vertices beyond those immediately adjacent to the NMV.

3.3.2 Decomposition along Non-manifold Edges

A non-manifold edge (NME) is identified by having both its end vertices to be of degree 4 or higher. At a vertex where the degree is 4, the NME has to be collinear with one other incident edge as well. Decomposition along a NME is similar to decomposition at a NMV. We also need to find all the vertices adjacent to the two endpoints of a NME. For example, in Fig. 3.5, the NME is marked with a bold line and its adjacent vertices are marked with circles.

Fig. 3.5. A non-manifold edge (in bold) and its adjacent vertices.

The basic idea for this decomposition is similar to that of Algorithm 3.1: vertices are in the same manifold if there exists a path connecting them that do not contain a vertex of any NME. We refrain from listing this algorithm as it is identical to Algorithm 3.1; instead of putting NMVs in the set U , we now put the end vertices of the NMEs.

Again, the complexity of the algorithm is $O(n)$, since it visits each of the n vertices only once. Fig. 3.6 shows two examples of decomposition along non-manifold edges.

Fig. 3.6. Two line drawings with non-manifold edges and their decomposed simple manifolds.

For drawings with a mixture of NMVs and NMEs, we apply the NMV and NME algorithms in sequence, simply by first putting all the NMVs into the set U and invoke the NMV algorithm, and then putting the end vertices of the NMEs into U and invoke the NME algorithm.

3.3.3 Decomposition across Internal Faces

To perform decomposition across internal faces, the methods of Chen et al [3] and Liu et al [44] require the detection of the internal faces, for which it needs to find all the faces first. Face finding is not a trivial problem, and is best avoided in the process of decomposition.

An internal face is generated by gluing two real faces, say f_1 and f_2 , together [44]. Internal faces can be divided into two types: (1) f_1 and f_2 do not share vertices, (2) f_1 and f_2 share some or all their vertices. If an object contains the first type of internal face, it must describe two or more disjoint edge-vertex graphs. Therefore, the decomposition along this type of internal face is trivial, as the groups of vertices are already disjoint. In this chapter, we focus on the decomposition of the second type.

In the rest of the chapter, for convenience of expression, a vertex with degree larger than 3 is called LV (L for larger and V for vertex); a vertex with degree less than 3 is called IV

(for incomplete vertex). An IV is the result of early-stage decomposition when it produces manifold and non-manifold components. A decomposed part containing an IV is not a manifold, and thus is an incomplete part. One that contains no IV is a manifold, a complete part.

Decomposition across internal faces is much more difficult than at non-manifold vertices or edges because, unlike a vertex or edge, a face is not directly available in line drawing. We do not have face information, let alone internal faces. Our strategy to decompose across an internal face is to decompose at vertices on the boundary of the internal face.

There are two stages: basic decomposition and repair for incomplete parts. Not all drawings require both stages; after the basic decomposition, there might be no incomplete part and hence repair is not required.

3.3.3.1 Basic Decomposition

In this stage, we are going to decompose complex manifolds that cannot be decomposed into constituting simple manifolds at non-manifold vertices or edges into the simple manifolds. We note that a decomposition may be non-unique, but for now, we will simply be seeking a legitimate decomposition and address the non-uniqueness issue later.

The decomposition strategy is to remove a simple manifold or a part thereof from the drawing at a time. The process is repeated until the remaining drawing belongs to only one simple manifold. Hence decomposition involves collecting all the vertices that belong to one simple manifold into a set, together with their connecting edges. We start by stating four properties which form the basis of our decomposition algorithm:

Property 3.1: A trihedral vertex belongs to only one simple manifold [15].

This is true except when two simple manifolds join at two overlapping faces that do not share their edges completely. But they would belong to only one simple manifold upon decomposition.

Property 3.2: A vertex connected with a vertex known to belong to only one simple manifold must belong to the same simple manifold.

If a vertex belongs to only one simple manifold, all the edges connected to it must belong to that simple manifold, which means all the vertices at the other ends of those edges must belong to the same manifold.

Property 3.3: As a partition must be made along exiting edges, these edges must form a loop. Such a loop bounds an internal face, which may be planar or non-planar.

Property 3.4: A vertex of degree 4 or greater (an LV) may belong to an internal face and may therefore border two simple manifolds.

Our method to collect all the vertices that belong to the same manifold can be summarized in Algorithm 3.2.

Algorithm 3.2: Forming of a simple manifold.

- 1) *Start with a vertex (call it start vertex) belonging to only one simple manifold*
- 2) *Add it and all its adjacent vertices to an empty set (Property 3.2).*
- 3) *For each vertex in the set*
- 4) *If its degree is 3 or less*
- 5) *Add its neighboring vertices to the set as they must belong to the same simple manifold. (Property 3.2 again)*
- 6) *Else the neighbors are not added to the set, which means disconnection occurs at the vertex.*
- 7) *When no more vertices can be added to the set, the part containing the vertices already added is decomposed from the original line drawing.*

This method may produce a part that is not a complete manifold. An LV may or may not belong to more than one simple manifold. Not knowing which the case is, we therefore do not collect the vertices connected to an LV. This means that decomposition may happen at an LV belonging to only one simple manifold. This LV then ends up in two decomposed parts. Vertex *a* in Fig. 3.7a is an example; it is an LV belonging to only one simple manifold. If we decompose the top part first, decomposition across it leads to two parts shown in Figs. 3.7b and c. Both parts contain IVs, therefore, they are incomplete. So, the existence of an LV belonging to only one simple manifold may result in incomplete parts.

There are two situations by which incomplete parts may be formed; we call the above Situation 1. Situation 2 is more complex; it arises when the face at the decomposition of a part is not a single but multiple connected internal faces. For example, decomposing the top part of the drawing in Fig. 3.7d will result in the incomplete top shown in Fig. 3.7e because there is no direct (i.e. single edge) connection between the two end points marked by circles and the two marked by squares due to the intervening vertices marked by dots. Incomplete parts can be repaired; this will be discussed in the next section.

Fig. 3.7. Two situations that result in incomplete parts.

The basic decomposition has three steps: *start vertex picking*, *vertex adding* and *result confirmation*.

Start vertex picking: A start vertex should belong to a single simple manifold only, which is the case for vertices of degree three or lower. This can be ensured by making the start vertex a vertex of the lowest degree in the current drawing. When both trihedral vertices and IVs are present, an IV has a higher priority because parts containing IVs have dangling components which need to be decomposed from the remaining drawing first. In the case where all the vertices in the remaining drawings are LVs, we stop decomposing, unless there are such LVs (for example, isolated vertices which will be introduced later) that we know must belong to one simple manifold only.

Vertex adding: Here we collect the vertices which definitely belong to the same simple manifold as the start vertex into a set, which we shall call $S1$. For this we use Property 3.2. As noted earlier, collection stops at an LV. After adding all the qualified vertices to $S1$, we put those vertices in the original line drawing but not in $S1$ in another empty set, $S2$. An LV that has been added to $S1$ may also belong to other manifolds. So if an LV has at least one of its adjacent vertices in $S2$, it is added to $S2$ also. After that, the original line drawing has been decomposed into two simpler parts.

Result confirmation: The two simpler parts are $S1$ and $S2$. $S1$, by the very nature of the decomposition, is either a simple manifold or a part of one and does not require further decomposition, but $S2$ may. The edges between the vertices in $S1$ and $S2$ are inherited from the original drawing. The degree of each vertex can thus be computed.

Let C be the current drawing we are decomposing, it may have zero or more decompositions before it. C may contain multiple vertices of the lowest degree which can serve as the start vertex, and different start vertices may result in different decomposed parts. We prefer to have $S1$ as a complete simple manifold, as this reduces the subsequent repair.

If the start vertex is an IV, $S1$ must be incomplete since it contains the start vertex. We then proceed immediately to decompose $S2$ if it needs further decomposition.

If all the available start vertices of C are trihedral, we can only arbitrarily pick one of them as the start vertex. At the end of the decomposition, if there is no IV in $S1$, then $S1$ is saved as a complete simple manifold, and we proceed to decompose $S2$. As we mentioned earlier, we prefer to have $S1$ as a complete simple manifold, but different start vertices may result in different decomposed parts, some of which may be incomplete. Therefore, if there are IVs in $S1$, we go back to try to decompose C by picking another start vertex that has not been used in the previous tries. The attempt is repeated until there is no IV in $S1$ or there is no more start vertex available.

Meanwhile, we save the decomposition result of each attempt. If $S1$ still contains an IV after trying all the start vertices available, then we keep the result of one of the attempts which meet the following requirement: the combination of the current decomposition

result and the part decomposed immediately before C contains the least number of IVs. This ensures that when no more complete part can be decomposed from C , those incomplete parts that belong to the same simple manifold are most likely to be decomposed successively. This is because, in general, the combination of two parts that belong to the same simple manifold is more likely to be complete than two parts that do not. If the current decomposition result is the very first decomposed part, in other words, C is the original drawing, we keep the decomposition result of any one attempt since there is no part decomposed before C . Then, we move to the next decomposition.

In short, the basic strategy of this step is, if there are multiple parts that can be decomposed from the current drawing, we should attempt to decompose a complete simple manifold first. If we have to decompose an incomplete part, the decomposition should produce as few IVs as possible.

Upon the completion of a decomposition, the part in $S1$ does not need further decomposition because it must be a simple manifold or is incomplete. The part in $S2$ is a simpler drawing than the original; we only need to further decompose this part by repeating the process.

Here, we use an example in Fig. 3.8 to illustrate how the basic decomposition is done.

We are interested only in the topology of the object and not the geometry.

Fig. 3.8. A line drawing of a complex manifold.

We pick a trihedral vertex as the first start vertex as they are vertices of the lowest degree in the drawing; there are ten such vertices and we can pick anyone of them. Assume that we pick Vertex 0. Then in the *vertex adding* step, we add it and all its adjacent vertices to an empty set $S1$, as they must belong to the same simple manifold, S . Now, vertices 0, 1, 3 and 7 are in $S1$. Then, we further add to $S1$ all the vertices that must belong to S . This means there are now eight vertices in $S1$: 0, 1, 2, 3, 4, 5, 6 and 7 (see Property 3.2). No more vertices can be added because vertices 4, 5, 6 and 7 are LVs. Then, we put the vertices in the original line drawing but not in $S1$ into another empty set $S2$. Check each LV that has been added to $S1$; add it to $S2$ if at least one of its adjacent vertices is in $S2$. So, $S2$ contains 12 vertices: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15. Note that the edge existing between two vertices in a set follows the vertices into the set. Then, we check that the two decomposed parts as shown in Fig. 3.9a are manifolds; both are simpler than the original line drawing. We then proceed to further decompose the drawing in $S2$.

Fig. 3.9. Two different decompositions of the line drawing in Fig. 3.8.

Alternatively, if we pick Vertex 8 as the start vertex, we would obtain the two parts as shown in Fig. 3.9b; one contains vertices 8, 4, 9, 11, 7 and 10, the other contains vertices 0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 12, 13, 14 and 15. In the first part, there are four IVs: vertices 4, 7, 9 and 10. Since this decomposition results in IVs, we would try other start vertices and

redo the decomposition. In this example, after changing to another initial vertex, such as 0 or 1, the original line drawing can be decomposed into two simpler manifolds.

Although most objects contain trihedral vertices, there are some which have none; for example, the objects in Fig. 3.10a and b. Such objects may contain LVs that only belong to one simple manifold. But they cannot be differentiated from the LVs that are on internal faces. There are two properties we can exploit to detect some of the vertices belonging only to one simple manifold.

Property 3.5: If the edges connecting the LVs (must be LV, because only LV may belong to multiple simple manifolds) adjacent to a vertex form a loop, then the vertex is isolated. The vertex and all its adjacent vertices can be decomposed as a simple manifold.

Property 3.6: If the edges connecting the LVs (also must be LVs) adjacent to an edge form a loop, then the edge is isolated, the edge and all its adjacent vertices can be decomposed as a simple manifold.

For example, the top and bottom vertices in Fig. 3.10a as well as the top and bottom edges in Fig. 3.10b possess such properties. Such vertices or edges are said to be isolated because they are isolated from other parts by the loop formed by the edges connecting their adjacent vertices. In addition, the loop formed by the edges connecting the adjacent vertices of an isolated vertex or edges does not have to lie on a plane face; i.e., it may self-intersect (such as the top bold in Fig. 3.10a). The vertex i in Fig. 3.10c is not an isolated vertex because, among its adjacent vertices, there are two vertices (marked by circles) with degree three. To perform a basic decomposition, we need to find isolated entities first and decompose them and all their adjacent vertices from the current drawing.

Fig. 3.10. Examples illustrating isolated edges and vertices.

The basic decomposition algorithm is summarized as follows:

Algorithm 3.3: Decomposition across internal face.

- 1) *Store the current line drawing in a set C*
- 2) *Find an isolated entity in C and decompose it with all its adjacent vertices as a simple manifold; repeat this step until no more isolated entity exists in C*
- 3) *Execute Algorithm 4.2 to decompose a simple manifold, all qualified vertices are stored in set $S1$*
- 4) *Save vertices in C but not in $S1$ to another empty $S2$*
- 5) *Check each LV in $S1$, if at least one of its adjacent vertices is in $S2$, add it to $S2$*
- 6) *If the start vertex is trihedral, keep the decomposition results according to the results confirmation*
- 7) *Save vertices in $S1$ as a decomposed part and clear $S1$*
- 8) *Replace vertices in C with vertices in $S1$, and clear $S2$*
- 9) *Compute the degree for each vertex in C and obtain the connectivities among the vertices*
- 10) *If there is no more LV in C , stop the decomposition*
- 11) *Else go to step 2*

If all the LVs in a line drawing are on internal faces and the decomposition is across a single internal face, then the drawing can be decomposed into multiple simpler drawings of manifolds completely. Two examples of such line drawings are shown in Fig. 3.11.

Fig. 3.11. Two line drawings [4], [44] and their decomposed simple manifolds.

In decomposing a part, the algorithm only visits each vertex once. The complexity of each decomposition attempt is therefore $O(n)$, where n is the number of vertices decomposed. But there may be multiple choices for the start vertex, say m of them, and in the worst case, everyone is tried in obtaining the desired result. The number of visits to vertices therefore $\sum_{i=1}^m n_i$, where n_i is the number of vertices visited at each attempt. If there are k simple manifolds to be decomposed, then the total number of vertices visited is $\sum_{j=1}^k \sum_{i=1}^{m_j} n_i$, where m_j is the number of available start vertices for each simple manifold. If m and n have similar values for every decomposition, then the worst case complexity is simply $O(kmn)$.

3.3.3.2 Repair of Incomplete Parts

Recall that, besides drawings that contain dangling edges and faces, an incomplete part may arise in two situations: 1) the line drawing contains LVs which are not on internal faces; 2) decomposition takes place across a series of connected faces. Incomplete parts need to be repaired to form manifolds.

Note that although both situations will result in incomplete parts, only the first one may result in an IV in the remaining drawing, as Fig. 3.7 illustrates. Given an incomplete part, we perform a step which we call P1.

Step P1: If the incomplete part resulted from an IV as the start vertex, combine it with its immediately previous decomposed part and save the combination as a new part.

An IV as the start vertex means that disconnection happened at an LV belonging to only one simple manifold during the decomposition of the immediately previous part. The adjacent vertices of the IV in the remaining drawing were adjacent to the LV in the original drawing. Therefore, the part that takes the IV as the start vertex should share the same simple manifold with the immediately previous decomposed part. Remember that IVs have the highest priority to be selected as the start vertex during basic decomposition. Therefore, these two parts that share the same simple manifold are decomposed successively.

Fig. 3.12. An example to demonstrate Step P1.

Take the line drawing in Fig. 3.12a as an example. The top part contains an LV, Vertex *a*, that does not lie on an internal face. If the part was decomposed first, the decomposed part and the remaining part would be as shown in Fig. 3.12b and c respectively. There is an IV (marked with a black dot) in the remaining part in Fig. 3.12c. This IV has the highest priority as the next start vertex, resulting in the triangle in Fig. 3.12d to be

decomposed from the remaining drawing. Step P1 combines the part in Fig. 3.12d with the part in Fig. 3.12b to form the new part shown in Fig. 3.12e. Note that the two edges in the base missing in Fig. 3.12b and are brought back automatically into the part in Fig. 3.12e because their end vertices are brought into the same part.

After executing Step P1, one thing can be guaranteed: the last decomposed part must be complete. This can be proved by contradiction. Assume that the last decomposed part is incomplete, which means that there is an IV in the remaining drawing after decomposing the second last part. Then, the start vertex of the last decomposed part must be an IV. Therefore, the last part will be combined with the second last part to form a new part according to Step P1. Then, the new part will be the last part. Once the start vertex of an incomplete part is detected as an IV, the incomplete part will be combined with its immediately previous decomposed part. For the last decomposed part, its IVs only result from the decomposition between it and its previous decomposed parts. Therefore, the last decomposed part must be complete after executing Step P1.

If all the decomposed parts are simple manifolds after the basic decomposition, we save them and the decomposition is complete. If incomplete parts exist, then we perform Step P1.

If there still exist incomplete parts after that, we save only the last part as a simple manifold and combine all the other parts into a new part, part N. N is simpler than the original drawing. Then perform Step P2.

Step P2: Execute the basic decomposition on N, then perform Step P1 on all its decomposed parts.

Every time we perform Step P2, we can decompose a simple manifold, the last decomposed part, from the drawing. Therefore, the drawing will become simpler and simpler by repeating Step P2, until part N cannot be decomposed further.

Fig. 3.13. An example.

Take the drawing in Fig. 3.13a as an example; it is to be decomposed into two simple manifolds shown in Fig. 3.13b. If the top part is decomposed first, the basic decomposition would produce three incomplete parts, as shown in Fig. 3.13c. Then the bottom part can be decomposed completely. Assume that the three incomplete parts in Fig. 3.13c are decomposed in the order of top to bottom. After executing Step P1, those incomplete parts still exist. Therefore, we save the last (bottom) part as a simple manifold and combine the other three parts into a new drawing, to perform the basic decomposition on it. But this new part is in face complete and cannot be decomposed further. Hence, we save it as a simple manifold. So, the original drawing is decomposed into two simple manifolds, as shown in Fig. 3.13b.

The incomplete parts produced by the basic decomposition and those produced by Step P1 have the same structure. Therefore, Step P2 can repair all incomplete parts resulting from both the situations we mentioned earlier.

Fig. 3.14. An example that cannot be decomposed.

There is a possibility that part N is incomplete. The drawing in Fig. 3.14a is such an example. In the basic decomposition, the trihedral vertex marked with a black dot can be used to start the decomposition, and the drawing would be decomposed into two parts, as shown in Fig. 3.14b and c. The part in Fig. 3.14c is the last one, and it is complete, since all its vertices are of degree 3 or higher. Although the part contains an LV, it cannot be decomposed further. Hence, we save it as a simple manifold. The part N in Fig. 3.14b is incomplete. Therefore, we cannot save it as a simple manifold. In this case, we perform Step P3:

Step P3: combine the part N with simple manifolds that share vertices which are IVs in N until there is no IV in N.

The combination will be saved as a simple manifold and those simple manifolds that have been combined with N will be deleted. So, for the drawing in Fig. 3.14a, finally, we get only one simple manifold which is the same as the original drawing. This example illustrates the situation that a drawing may not be decomposable, but we have to perform the algorithm in order to ascertain that, wasting a little time in the process.

The whole algorithm for decomposing and repairing a drawing is summarized as follow:

Algorithm 3.4: the whole algorithm.

- 1) *Perform the basic decomposition*
- 2) *Check all decomposed parts*
- 3) *If there is no incomplete part*
- 4) *Save all parts as simple manifolds and finish.*
- 5) *Else*
- 6) *Perform Step P1*
- 7) *Save the last decomposed part as simple manifold*
- 8) *Combine all parts but the last one into a new drawing, N*
- 9) *If N can be decomposed further*
- 10) *Perform Step P2, clear N and go to step 7*
- 11) *If N is complete*
- 12) *Save it as a simple manifold and finish*
- 13) *Else*
- 14) *Perform Step P3*

3.3.4 A Special Case: High Degree Manifold Vertices Forming a Loop

A special situation arises when manifold vertices of degree 6 or higher in a drawing form a loop that encloses an internal face. Partition occurs when this drawing is processed by the NMV Algorithm, even though there are no non-manifold vertices. That is because vertices of degree 6 or higher are assumed to be non-manifold vertices initially. In such a situation, the NMV Algorithm actually performs the task of the Internal Face Algorithm, and decomposes the object across the internal face.

Fig. 3.15a shows an example of a manifold object with eight degree-6 vertices that form six internal faces. None of the vertices are non-manifold. But because they are of degree 6, they are taken as non-manifold vertices and the NMV Algorithm is invoked first. Normally there is no decomposition at a degree-6 vertex that is not non-manifold. But in this case, because sets of four such vertices form loops of internal faces that separate one simple manifold from another, the NMV Algorithm decomposes one simple manifold upon the first application. Assume that the simple manifold connected to the loop *abcd*a

is decomposed, the remaining drawing still contains four vertices of degree 6, e, f, g and h . So the NMV Algorithm is applied again, and this time the simple manifold connected to loop $efghe$ is decomposed. The result is shown in Fig. 3.15b, with the cross remaining to be decomposed further. This cross has eight vertices of degree 5. So the algorithm for NME is invoked, even though the object contains no non-manifold edge. As in the case of degree-6 vertices, this algorithm is applied twice, and Fig. 3.15c shows the outcome, with the two decompositions across loops $abfea$ and $dcghd$. The remaining bar with eight vertices of degree 4 is then decomposed using the Internal Face Algorithm to produce the final decomposition shown in Fig. 3.15d.

Fig. 3.15. The partition of an object with degree-6 vertices forming internal faces.

The whole drawing is therefore decomposed correctly, even though both the NMV and NME algorithms are employed for manifold vertices. This special situation of high degree vertices forming loops of internal faces is the only one where the two non-manifold algorithms perform decomposition on manifold objects, with correct outcomes.

3.4 Results

We implemented the algorithm and all subsequent algorithms in this thesis on a PC with Intel Core™ 2 Duo CPU E6750 @ 2.66GHz in C++. Fig. 3.16 shows a set of test objects successfully decomposed into simple manifolds.

Fig. 3.16. A set of examples and their decomposition with the computational times in milliseconds. Drawings e, g, h, j are from [44] and drawing i from [4], drawing k demonstrate the special case of Section 3.3.4.

To show that decomposition will significantly reduce the computational time for the subsequent face finding process, we used an example first used in [44] (Fig. 3.17) to compare the time consumed for a line drawing for two processes: 1) decomposition first, followed by face finding; 2) direct face finding without decomposition.

Fig. 3.17. Seven line drawings of increasing size denoted by seven loops [44].

Table 3.1 Time comparison between decomposing plus face finding and direct face finding for drawings in Fig. 3.17. The times in the last column are from [44].

	Time for decomposition	Time for face finding after decomposition	Time for directly finding internal faces
1	0.1	1.6	138
2	0.5	2.8	1445
3	1.9	3.8	3642
4	3.9	4.9	7830
5	5.4	6.0	13142
6	10.5	7.2	17505
7	18.8	8.3	22712

In Table 3.1, the first column shows the consumed time for decomposing line drawings in Fig. 3.17. The second column shows the computational time for face finding after decomposition, where the face finding algorithm is that of Liu and Lee [45]. The last column shows the time for finding internal faces by Liu et al. [44]; the times for the face finding must be larger than those shown in the column, because the times for finding all

the faces must be larger than that for finding only the internal faces. Their face finding algorithm is that of Liu and Tang [46].

Table 1 shows that of our method is much more efficiency than that of [44]. Also, it will result in simpler and more reliable face finding and 3D reconstruction, as these processes have high complexity over the number of vertices in the drawing.

Earlier, we mentioned the non-uniqueness issue in decomposing across internal faces, resulting in multiple solutions in internal face (planar or non-planar) identification. Fig. 3.18a shows an example. The object has three internal faces, bold loops in Fig. 3.18b, c and d. Thus, there are three options to decompose the object, which bring about three sets of simpler manifolds and all of them are correct. In this case, our method is able to return every of the three decompositions; which is deliver, depends on the start vertex. Vertex i , j or k as the start vertex give the results shown in Fig. 3.18b, c and d respectively.

Fig. 3.18. An object and its three options for decomposition.

3.5 Chapter Summary

This chapter introduces a novel method to decompose line drawings depicting complex manifolds into ones of simpler manifolds, directly without face finding. Our method performs decomposition at non-manifold vertices and/or edges and across internal faces

successively if necessary. Decomposition along non-manifold vertices and edges are relatively simple. But decomposition across internal faces is not, since we do not have face information. The method for decomposition across internal faces has two stages: basic decomposition to collect edges and vertices that must belong to the same simple manifold followed by repair, which returns incomplete decomposed parts as correct complete manifolds. A complex line drawing may not be decomposable into simple manifolds, but if it can be, then the decomposed entities must be manifolds simpler than the original line drawing.

Finding the faces of a complex manifold in a line drawing has high computational complexity. By decomposing the drawing into a collection of drawings of simple or simpler manifolds would reduce the computational cost greatly. The crux of the matter lies in the fact that our algorithm for decomposition is very efficient. Its cost is therefore small, and in general is a lot less than the savings that can be achieved in face finding. Furthermore, the downstream process of 3D object reconstruction benefits greatly too from the decomposition. In the worst case, when the decomposition process results in no decomposition, which means that there is no saving in face finding or 3D reconstruction, this additional cost is so small that it would not unduly affect the overall performance.

CHAPTER 4

A New Face Identification Method by Drawing Decomposition

4.1 Introduction

The decomposition algorithm in the previous chapter led us to think that it might be possible to find faces by decomposition too. After all, faces are but a finer partition of a drawing.

This chapter presents the result of the investigation that ensued, which is a face-finding method from single line drawings that is totally different from existing ones. It consists of two stages: decomposition and face forming, which parallel the decomposition and repair of the previous decomposition algorithm. Here, a drawing is decomposed into chains of connected edges. Each chain belongs to one face only, but may not yet form a closed loop. The decomposition is local to a vertex, and takes no account of the overall structure of the drawing. The algorithm ensures that there is a chain for every face in the object. The face forming stage completes the loop for each chain to form a face. Our method can deliver both the real and the internal faces separately. We implemented our algorithm and tested it with hundreds of objects, including objects used in previous publications. It always delivers the correct result. The time comparison with the previous methods shows that it is faster often by tens of times, and in some cases, hundreds of times.

For ease of reference, we call this new method DFF, for decomposition and face forming.

DFF handles manifolds and non-manifolds. But we first focus the presentation of the algorithm on manifolds, as the treatment for non-manifolds, which will be dealt with later, is largely the same.

4.2 Related Work

There are numerous algorithms for finding faces in line drawings. Markowsky and Wesley [47] presented a method that requires 3D coordinates to find normal of planes which are determined by two non-collinear adjacent edges of a vertex. They then tried to find the other vertices which are located on each plane. They also discussed some special issues involved in face identification, such as the internal face problem, and proposed a solution to avoid it. Their method can only handle objects with planar faces.

Hanrahan [48] and Dutton and Brigham [49] used topological rather than geometrical information to identify faces of a drawing. Their algorithms are only able to handle objects without holes whose graphs are 3-connected, but the drawings of many objects are not 3-connected. Granter and Uicker [50] identified possible cycle sets based on finding a minimum spanning tree. They also proposed two properties to select the most possible faces. Their method is not able to deal with objects containing holes and internal faces. Courter and Brewer [51] improved the method by introducing a better procedure to select faces from cycles. Their algorithm can detect internal faces automatically but cannot handle objects of genus larger than 0.

Bagali and Waggespack [52] used an efficient shortest path algorithm to generate cycles. Their method can find faces very fast, but the limitation is that it can only handle objects whose drawings are 3-connected.

Shpitalni and Lipson [53] proposed two methods for face identification, one for objects of genus 0 and another for general objects. Their method finds all the potential faces first, and then uses a heuristic search technique, based on the observation that a human tends to select a face configuration in which as many edges as possible take part in as many faces as possible, to find the actual faces. Their method cannot handle objects with internal faces. Liu and Lee [45] revised the algorithm of Shpitalni and Lipson, they first find all the possible face loops and then use several properties to determine the actual faces. Their method can handle both loop ambiguity and internal faces, which earlier methods cannot. But the time complexity of their algorithm is $O(e^v)$, where e is the base of natural logarithm and v is the number of vertices in an object. Based on the work of Liu and Lee, Liu et al. [54] proposed a new algorithm: use a depth-first search scheme to search all the possible cycles first, so that those cycles that are real faces can be found more easily in the second step. Compared with the algorithm of [45], this method makes significant improvements, as the depth-first search scheme can make correct decisions early. It can handle objects with planar and curved faces. But the basic problem still exists - the number of cycles produced in the first step grows exponentially with the number of edges of a line drawing.

Liu and Tang [46] proposed a genetic algorithm (GA) to find faces from 2D line drawings. To solve the problems that appeared in the work of Liu et al. [54], they used a hybrid GA (HGA) by combining the standard GA with a novel local search scheme. The GA is good at local searching and the novel local search scheme has a high probability in finding faces directly from the drawing. Their results demonstrate that the HGA is robust in finding optimal solutions. But they cannot guarantee that their algorithm is repeatable.

The algorithm by Varley and Company [55] searches a graph for loops; it imposes context-dependent path length as the traversal cost to find the least-cost closed loops. Initially, it divides an edge into two half-edges with opposite directions, and uses them to form strings of ordered sequence of two or more vertices. Starting from an arbitrary trihedral junction, one half edge forms a string towards the junction and the other leaving the junction. Two strings are concatenated if the last vertex of one is the first vertex of the other. If two strings form a closed loop, they are merged to form a face. The algorithm tends to find the smaller faces which are more likely to be the correct ones. The algorithm is more efficient than those reported previously. But it works only on manifolds, and as reported in [55], it fails occasionally.

The work of Du et al [56] is an improvement on the method of Liu et al [54], which uses depth-first search to find the cycles in a graph. The number of cycles can be very large, and it exploits a set of properties of faces in manifolds to reduce the possible cycles, eventually leading to the actual faces of the object. Du et al improve the process by taking account of hidden lines in the drawing to further reduce the number of cycles which are candidates for faces.

Leong et al. [6] introduced an innovation, which is a mechanism for validating the faces found. Although other algorithms aim to deliver the correct results, they cannot guarantee that they do. It is up to the user to check. This is a problem when face finding is a part of an automated process, such as 3D shape reconstruction from line drawings. The validation in [6] determines if the set of faces found is correct, and it has built-in ability to auto-correct if a result fails the validation. There are five stages in the correction, and if none of them produces a valid result, an error message is returned. The face finding

algorithm itself starts from an edge and performs two parallel breadth-first searches, one from each end vertex of the edge. When the two search trees meet at a vertex, a closed loop, and thus a potential face, is found. Breadth-first search produces the smaller loops first, which are more likely to be actual faces. This method works for both manifolds and non-manifolds, and is efficient for most drawings.

The existing methods have one common characteristic: the search for a face potentially requires traversal of the whole drawing, leading to long searches and high complexity. But a face is a rather local entity with vertices of degree 2. We can look to exploiting these simple facts to construct a very efficient algorithm.

In this chapter, we introduce a new algorithm which does that. It forms a part of every face first, and uses that to search for the missing part, within a very limited search scope. This is the basis of its efficiency.

4.3 Terminologies

Specialized terms that appear in this chapter are summarized in this section.

Sub-loop: A series of connected edges that potentially forms a part of the loop of a face. The vertices in a sub-loop are of degree 2 except at the two ends where the degree is 1.

Active sub-loop: An active sub-loop is a sub-loop that is being processed to form a face.

Current drawing: The current drawing is the one that is being decomposed.

Remaining drawing: The remaining drawing contains the remaining edges of the current drawing after decomposing a sub-loop. If further decomposition is needed, the remaining drawing becomes the current drawing in the next decomposition.

4.4 The Preprocessing and the Main Algorithm

In the preprocessing, we split drawings consisting of a combination of manifolds and non-manifolds into their constituting components. Dangling edges, i.e. edges that do not form a loop, are detected at this stage and removed from the drawing, as they do not contribute to the formation of faces. We then find all the non-manifold vertices and non-manifold edges according to their definitions given by Sun and Lee [43]. If a line drawing contains a non-manifold vertex or edge which is a junction between components (see Fig. 4.1), it can be separated into the components at these junctions, and the faces for the components can be formed separately.

Fig. 4.1. Topologically non-manifold drawings that may be manifold geometrically.

A topologically non-manifold drawing may represent a non-manifold or manifold geometrically, as shown in Fig. 4.1. The first and third drawings are clearly non-manifold, while the second and fourth may be manifold geometrically. Topologically, however, Fig. 4.1a and 4.1b are the same, i.e., non-manifold, and likewise Fig. 4.1c and 4.1d. Our algorithm only deals with single manifolds, and hence the four figures would all be decomposed into two manifolds first before the application of the algorithm.

The decomposition algorithm of Chapter 3 can be applied on a complex drawing, reducing it to a collection of simple manifolds. The face finding algorithm can then be applied on these simple manifolds. However, in this chapter, we will develop the face finding algorithm on manifolds without decomposition, for completeness of the presentation.

Even though our algorithm applies to both manifolds and non-manifolds, we shall focus only on manifolds in the presentation that follows. That is because, as we shall see, the algorithm works for both identically, except for some details, which we will discuss separately. First, we shall give a quick preview of the algorithm using an example.

The basic idea is to decompose a drawing into separate chains of edges, with each chain belonging to one face. We call these edge chains “sub-loops”. Each sub-loop should contain only edges that can be ascertained to belong to the same face. After the decomposition, we form faces with the sub-loops. Take a simple rectangular block as an example (Fig. 4.2a); all its vertices are trihedral. We know that every edge pair at a trihedral vertex must belong to one face [54]. Hence we can form three sub-loops with the three edges, say at Vertex a , as shown in Fig. 4.2b. We note that only two edges are in each sub-loop; that is because there are two other edges available at each of the two free ends of the sub-loop, and we cannot determine at this stage which one of the two belongs to the same face as the sub-loop.

Fig. 4.2. The decomposition of a block.

Once formed, these sub-loops are stored separately and their edges removed from the drawing. Then we proceed to decompose the remaining drawing, deliberately at a vertex of the lowest degree, which is 2 in this case. There are three such vertices, b , d and e . Fig. 2c shows the decomposition at Vertex b , which again produces a sub-loop of two edges. This is followed by the decomposition at Vertex e , say, this time forming a sub-loop of three edges because Vertex f is also of degree 2 and the edges connected to it must belong to the same face as those connected to Vertex e . The remaining drawing is a complete loop $cdhgc$, which does not require decomposition. $cdhgc$ thus forms a face of the object.

The incomplete sub-loops, in the order they are formed, are bad , bae , dae , cbf and $hefg$. The first three are formed at the same time and the order among them is arbitrary.

The next task is to form the remaining faces from the sub-loops. We take the sub-loops in reverse to the order they are formed. Hence, $hefg$ is dealt with first, and its missing edge, hg , can be found in (and only in) the last complete loop $cdhgc$. Then comes the sub-loop cbf , and its two missing edges, cg and gf , can be found in the two loops formed thus far, $cdhgc$ and $hefgh$. Similarly, the missing edges of the other three sub-loops could all be

found in the loops formed beforehand. Note the limited search space for each sub-loop; this is important for the efficiency of the algorithm. Once we have visited all the sub-loops, the face forming is complete.

The whole process can be summarized in the following algorithm:

- 1) *Starting from a vertex of the lowest degree, decompose input drawing into sub-loops*
- 2) *Starting from the last incomplete sub-loop in reverse order to the decomposition*
- 3) *Form face with each sub-loop*

This algorithm is intuitively simple at a high level. But the example leaves out many details and raises many questions. These will be addressed in the sections that follow.

4.5 Decomposition

4.5.1 Decomposing a line Drawing into Sub-loops

In this stage, a line drawing is decomposed into sub-loops, each containing at least two connected edges that belong to only one face. The strategy is to find simple rules that determine that two or more edges can belong to the same face, and use these rules to effect the decomposition. Given that we are dealing with manifold objects only, all the vertices are of degree at least 3 initially. However, when decomposition is in progress, there may be vertices of degree 2 or even 1. We define four rules, D1 to D4, where D stands for decomposition, to facilitate the decomposition.

Rule D1: Always start forming a sub-loop at a vertex of the lowest degree.

The reasons for this rule will become apparent as we progress.

Rule D2: Terminate a sub-loop at a vertex of degree 3 or higher.

This is because we do not know which of the multiple adjacent edges at the vertex belongs to the same face. This results in an incomplete sub-loop. Upon termination, the sub-loop is stored separately and its edges are removed from the drawing.

Rule D3: Within a face, every vertex has degree 2. If a vertex has degree 2 in the drawing, its two adjacent edges must belong to the same face, so collect them into one sub-loop.

To explain this rule, we first pose a conjecture.

Conjecture 1: By virtue of Rules D1 and D2, we will never get into a situation where edges that do not belong to the same face are left in a chain connected by vertices of degree 2.

This is a conjecture because we have no formal proof. But we have no counter example either. For example, in Fig. 4.2, it is not possible to have edges bc , cg and gh left in a chain with the connecting vertices c and g at degree 2. One of the two end vertices b and h would have been removed by an earlier decomposition while one or both of c and g were at degree 3. Therefore adjacent edges whose connecting vertices are of degree 2 must belong to the same face and are thus collected into the same sub-loop, forming a multi-edge chain. That is the case for sub-loop $hefg$ in Fig. 4.2d.

Rule D4: A vertex of degree larger than 2 can be shared by multiple faces, but it may not be possible to determine at this initial stage which two incident edges belong to a real face, hence create a sub-loop for every edge pair.

If the start vertex (call it A) is of degree 3 or greater, then every vertex at the other ends of the edges adjacent to A must be of degree 3 or greater as well since we always start at

a vertex of the lowest degree. Hence, the sub-loops obtained here can only contain two edges immediately adjacent to A, due to Rule D2.

In manifolds, every vertex of degree 3 must be shared by three faces, and every pair in the three edges must belong to one face [54]. For vertices of degree larger than 3, there is no certainty at this early stage which two edges border a real face. So every pair is taken as one sub-loop. If the degree of the vertex is n , we will get C_n^2 new sub-loops. There are redundant sub-loops here; this will be dealt with during face forming.

Revisiting the block in Fig. 4.2a, all its vertices are of degree 3; decomposition can start with anyone of them. Fig. 4.2b shows starting at Vertex a , forming three new sub-loops bae , dae , and bad ; they will all form real faces. But this is not true for a vertex with degree larger than 3. For instance, the degree of all the vertices of the drawing in Fig. 4.3a is 4. If we select vertex m as the start vertex, six sub-loops, pms , qms , rms , pmr , qmr and rms can be obtained. pms and qmr cannot form real faces, but that cannot be determined at this stage. So we simply retain all six as sub-loops. Those that do not form faces will be eliminated later in the face forming stage. The decomposition at Vertex m results in four trihedral vertices in the remaining drawing. They are now vertices of the lowest degree and hence one of them would be the next start vertex. Even though the degrees of the vertices in the drawing are all 4 initially, the degrees of the start vertices for the subsequent decompositions are all less than 4. Hence they produce no redundant sub-loops, though some of them may belong to internal faces.

Fig. 4.3. A drawing with all its vertices of degree 4.

Redundancy occurs only when every vertex in a drawing is of degree higher than 3. Such a drawing, however, may be the result of a decomposition, such as the decomposition at Vertex t of degree 3 in Fig. 4.3b, which leaves a drawing whose vertices are all of degree 4. Reducing redundancy is one reason for always starting a decomposition at a vertex with the lowest degree. In other cases where the original manifold contains a mixture of vertices of degree 3 and higher, there is no redundancy at all. We prove that in Theorem 4.1.

Theorem 4.1: No redundant sub-loops can be created apart from a decomposition at a start vertex of degree greater than 3.

Fig. 4.4. A drawing with a vertex with degree of 4.

Proof: We first prove the case for a pyramid, shown in Fig. 4.4, where there is one vertex of degree 4, Vertex a , and two possible redundant sub-loops, bad and cae . Note that the redundant sub-loops must have Vertex a as the connecting vertex. Because Vertex a is not a vertex of the lowest degree, it cannot be the start vertex of the first decomposition.

So the redundant sub-loops cannot be formed then. By starting the decomposition at a vertex of the lowest degree and because sub-loop formation stops at vertices of degree higher than 2, Vertex a can only be an end vertex of a sub-loop, until it is reduced to a vertex of degree 2. But by then, we are left with a single loop which does not contain either of the redundant sub-loops. For example, starting from Vertex b , say, produces three sub-loops, cba , cbe and eba , reducing the degrees of Vertices c and e to 2. So one of them must be the next start vertex, leading to the sub-loop dca or dea . Either way, it leaves a triangular loop containing Vertex a but no redundant sub-loop.

The same procedure will show that no redundant sub-loops can be formed at vertices of higher degrees in the presence of vertices of degree 3. ■

It is important that decomposition does not result in faces being lost; that is, every face of the object in a drawing must have a sub-loop associated with it. We prove it in Theorem 4.2.

Theorem 4.2: Every real face in a drawing is associated with a sub-loop.

Proof: Rule D4 guarantees that after the decomposition at a vertex, the faces sharing the edges connected to this vertex would be fully accounted for by the sub-loops formed then. No other faces in the remaining drawing can have any of these edges in their boundaries. Therefore the removal of these edges from the drawing upon decomposition would not result in the loss of any faces. So the remaining drawing would contain all the remaining faces, and could be treated like a new drawing and decomposed in the same way. ■

In fact, the sub-loops also account for internal faces, but not always all possible internal faces, as will be explained in Section 4.7.2.

The decomposition process reduces the size of a drawing progressively. Edges decomposed are removed from the current drawing and stored separately, upon which the decomposition starts again with the remaining drawing. The next start vertex may have a degree lower than the current one. After finishing the decomposition, most sub-loops would be incomplete, with one or more vertices of degree 2 and two of degree 1. If a sub-loop contains only vertices of degree 2, then it must be closed and is thus complete.

There are two issues that need special treatment in a decomposition.

- 1) A vertex of degree 1 may be created in a drawing after a sequence of decompositions.

We first demonstrate how this may arise using an example. Take the line drawing in Fig. 4.5a; it depicts a non-manifold, but may be arrived at through the decomposition of a manifold. There are eight vertices (marked with dots) that can be the start vertex. Assume that Vertex i is selected first, then the sub-loop in Fig. 4.5b is removed from the drawing. Then vertices j and k are selected successively, resulting in the sub-loops shown in Fig. 4.5c and 4.5d. The remaining drawing, with a vertex of degree 1, is shown in Fig. 4.5e. The lone dangling edge adjacent to this vertex cannot form a sub-loop with any other edges. But it is a part of a face and needs to be available when required. A special storage is therefore created to save this edge; it is a linear list which we call U , for unused. Save the edge in U and remove it from the current drawing. The remaining and last sub-loop is a complete loop. As we shall see later, U is also used to store other unused edges, from sub-loops with which faces cannot be formed immediately.

Fig. 4.5. An example of decomposing a drawing.

2) The lowest degree of all the vertices in the remaining drawing is larger than 2.

In this case, there may be an internal face in the drawing. Internal faces can exist only when there are vertices of degree larger than 3. Take the drawing in Fig. 6a, Loop $bcdeb$ is an internal face. Assume Vertex a is the start vertex for decomposition. The edges in bold would then be decomposed into five sub-loops. The vertices of the remaining drawing, shown in Fig. 6b, all have degree 3. This means that, prior to the decomposition, there must be vertices with degree larger than 3, which means an internal face exists between the component that has been decomposed and the remaining component.

We mark the existence of an internal face by noting the position of the last decomposed sub-loop in the list of sub-loops. If the previous sub-loop produced is the k th one, then we store the value k in a list. We call this list $\mathbf{I}$, for internal. In the example, the number 5 would be stored in $\mathbf{I}$ as the last sub-loop produced was the fifth one.

Fig. 4.6. A drawing which remains a manifold after some decomposing.

This position number in I divides the sub-loops into two groups, those decomposed before and those decomposed after that number. If there are n internal faces, then there are n numbers stored in I , and $n+1$ groups. This grouping is required to deal with issues arising from drawings of objects with holes, as we shall see in Section 4.6.3.

The decomposition algorithm can be summarized as follows:

Algorithm 4.1: the decomposition algorithm.

- 1) *While there are edges in the current drawing*
- 2) *Select a vertex with the lowest degree as the start vertex v*
- 3) *If the degree of v is one, move its adjacent edge to U*
- 4) *Else if the degree of v is 2*
 - 4.1) *Save the two adjacent edges of v and further edges connected at vertices of degree 2 as a new sub-loop*
- 5) *Else if the degree of v is larger than 2*
 - 5.1) *If v is not the very first vertex, put the position number of the last decomposed sub-loop in I*
 - 5.2) *Save every pair of the adjacent edges of v as a new sub-loop*
- 6) *Remove the edges involved from the drawing*
- 7) *End While*

During decomposition, one vertex is visited at most once, so the time complexity of the algorithm is $O(v)$, where v is the number of vertices in the drawing.

The number of sub-loops produced is always very close to the actual number of faces in the object, as the number of redundant sub-loops is minimized by the decomposition

starting at a vertex of the lowest degree. Potentially, at a vertex with n incident edges, the pairwise combination of edges numbers C_n^2 , which can be a lot higher than the number of real faces, which is also n . But we very seldom have to use vertices with a high number of incident edges as the start vertex, hence this is not a major issue.

4.5.2 Properties of Sub-loops

Sub-loops have a number of important properties.

Property 4.0. Every sub-loop contains at least two edges. The degree of a vertex in a sub-loop is at most 2.

This is the inherent property of a sub-loop required in the decomposition.

Property 4.1. Each sub-loop belongs to at most one face.

The decomposition process ensures that this is true. When we do not know which of the edges adjacent to a vertex shares the same face with a sub-loop, we do not add any of them to the sub-loop. But a sub-loop may not belong to any face.

Property 4.2. Multiple sub-loops may belong to the same face.

This may happen, especially when there are multiple vertices with the lowest degree in the current drawing. Looking ahead, sub-loops 1 and 2 in Fig. 8a belong to the same face.

Property 4.3. Each face must have at least one sub-loop that belongs to it.

This has been proven in Theorem 4.2. This property is important because it means that all the faces can be recovered by forming faces from the sub-loops that exist.

Property 4.4. *Some sub-loops may not be a part of any face.*

This is a result of the decomposition at a vertex of degree larger than 3. The redundant sub-loops will be eliminated in the face forming stage.

Property 4.5. *The missing edges of the face of a sub-loop only exist in the remaining drawing, and not in the sub-loops that have been decomposed previously.*

This is a fact of the decomposition process: sub-loops are decomposed from a drawing when the edges it can form a complete loop with are also in the drawing. Therefore, the missing edges of a sub-loop must exist only in the remaining drawing. Let us look at the block in Fig. 2a again. After the first decomposition at Vertex a , we get three sub-loops. The edges that can form a face with each of them are in the remaining drawing in Fig. 2b. This is true of the first decomposition, and is true of every subsequent decomposition as well, with one exception.

The exception occurs when multiple sub-loops belong to the same face (Property 4.2). These sub-loops are decomposed from the drawing one after another, though not necessarily consecutively. Obviously, the sub-loop formed later would have some of its missing edges in the sub-loop decomposed earlier, and not only in the remaining drawing.

Property 4.6. *The last sub-loop must be a complete loop.*

We prove this by contradiction. Assume that the last sub-loop is incomplete, then, it must contain two vertices with degree 1. There are two situations by which this may arise: (1) the two vertices already existed before decomposing the second last sub-loop or (2) they were produced upon decomposing the second last sub-loop. For the first situation, the

vertices with degree 1 cannot be in the last sub-loop because we always choose a vertex of the lowest degree as the start vertex for a decomposition. The vertices and their adjacent edges would have been decomposed earlier. As for the second situation, the two vertices with degree 1 must be the connections between the second last sub-loop and the last sub-loop, which means that the second last sub-loop was decomposed by disconnecting at these two vertices. But decomposition involves removing at most one edge at a connecting vertex, which means prior to the decomposition, the degree of the vertex is 2. This further means that the edges across this connecting vertex must belong to the same sub-loop. Disconnection at this vertex therefore contradicts the decomposition process and cannot happen. Hence, the last sub-loop must be complete.

In fact this property is a lot stronger, as stated in Conjecture 2.

Conjecture 2: *The last sub-loop must be a real or internal face loop or a hole loop.*

That means it cannot be simply any loop in a drawing apart from the above three, which have the same topology. For example, In Fig. 4.7a, it cannot be the silhouette loop shown bold. For a loop to remain as the last loop, none of its vertices can serve as the start vertex to any previous decomposition. Hence only Vertex a or b can be the first start vertex. Assume that it is Vertex b , then after the decomposition, we are left with the drawing in Fig. 4.7b, which has three vertices of the lowest degree, 2. These are all on the silhouette loop which must then be decomposed, and thus cannot remain as the last loop. It also cannot be the loop shown in bold in Fig. 4.7c, which contains a chord joining two vertices of degree 3. This loop must be further decomposed since a face loop can contain only vertices of degree 2. Similar situations would arise if we pick any loop that is not

one of the three types named above. As yet, we are not able to prove that only these types of loops can be the last loop but, as with Conjecture 1, we have no counter example either.

We can strengthen this property even further by a simple manipulation, so that the last loop is always a real face loop. If the last loop is an internal face loop or a hole loop, then it would always contain vertices initially of degree 4 or higher, whereas real face loops usually contain vertices of degree 3, except when the initial drawing contains no such vertex. When we arrive at a situation when there are multiple vertices of the same lowest degree available as the start vertex, we only select vertices which have the highest degree in the initial drawing. This means that internal faces and holes would be decomposed first before real faces, and thus ensures that the last sub-loop is a real face loop.

Fig. 4.7. Decomposition of a block

4.6 Face Forming

After the decomposition, we have a set of sub-loops and the last one is a complete loop (Property 4.6). We now proceed to form the faces from the sub-loops.

4.6.1 Forming of Individual Faces

Face forming complies with two rules, F1 and F2, F for face forming.

Rule F1: A face cannot share more than one non-collinear edge with another face. This is the face adjacency theorem [53].

Rule F2: An edge is shared by exactly two real faces. This is a property of manifolds [54].

The order of face forming is in reverse to the order of sub-loop decomposition. This reverse order exploits Property 4.5: given a sub loop, we only need to search the edges in the remaining drawing for the missing edges. This greatly reduces the scope of the search and thus improves its efficiency. For example, the first sub-loop for which we need to find missing edges is normally the second last in the decomposition sequence, since the last is always complete. The reverse order means that we only need to find its missing edges from the last sub-loop, which contains only a small number of candidate edges compared to the whole drawing.

A candidate edge is an edge available to complete the active sub-loop. Our algorithm maintains a candidate edge list, which we call *CE*; its members are added and deleted as faces are formed. At the start of face forming, the members in the list are only the edges of the last loop. Visiting the example of Fig. 4.2 again, the last loop is *cdhgc*, and so the edges in *CE* are *cd*, *dh*, *hg* and *gc*. The first active sub-loop for face forming is *hefg* and its missing edge, *hg*, can be found in *CE*. Once the loop *hefgh* is formed, *hg* is no more available as a candidate edge due to Rule F2 and is thus removed from *CE*. But three new candidate edges *he*, *ef* and *fg* are added. The next active sub-loop, *cbf*, requires edges *fg* and *gc* from *CE* to complete its loop. These two edges are then removed from *CE*, again due to Rule F2, but edges *cb* and *bf* are added. Thus members are added and removed from *CE* as face forming progresses, but their number is always a lot smaller than the

total number of edges in the drawing. Another source of members for **CE** is the list **U**, which contains edges that have not been used. A member, m , of **U** is placed in **CE** only when the sub-loop decomposed before m was decomposed becomes the active sub-loop. This is because, during decomposition, a lone dangling edge cannot form a face with any edge in the remaining drawing, therefore we put it in **U**. But it may form a face with sub-loops decomposed before it. A member needs to be removed from **U** once it is used to form face.

A sub-loop is flagged as “complete” if it is closed and “incomplete” otherwise.

In an incomplete sub-loop, there must be two vertices with degree 1. These two vertices are joints where the existing edge chain in the sub-loop would link up with the missing edges to form a complete loop, and thus a face. We denote one of these two vertices as the start point (**PS**) of the chain and the other as the end point (**PE**) of the chain.

Internal faces add complications to the face forming process. In this section, we introduce face forming for line drawings containing no internal faces. We will deal with internal faces later. We use depth first search to find the missing edge chain. Starting from **PS**, we find all the chains using the edges in **CE** based on the two F rules. Chains not ending at **PE** are discarded.

The number of chains that start at **PS** and end at **PE** for an active sub-loop is variable; there are three possible situations.

4.6.1.1 No Chain Found

When no valid chain is found, no loop can be formed with the active sub-loop. There are two causes. The first is a redundant sub-loop introduced at decomposition which cannot form a face at all. The second is when a loop was decomposed into multiple sub-loops. In both cases, the sub-loop is put in U , as it has not yet been used. The source of redundant sub-loops has been explained. We illustrate the second with an example using a non-manifold drawing.

Fig. 4.8. A simple example.

For the line drawing in Fig. 4.8a, assume that vertices 1 and 2 are selected as the first and second start vertices for decomposition. The two sub-loops generated are shown in bold, leaving behind two complete loops. Only the first two sub-loops need to be processed. Because the order of face forming is the reverse of the decomposition order, the second sub-loop is handled first. Starting from its PS , we find all the chains based on Rules F1 and F2. Two chains can be found, as shown in Fig. 4.8b: Chain 1 (with e_0) and Chain 2 (with e_1). The chain with e_0 cannot proceed further because e_0 and the next edge are both used by a complete sub-loop already. The chain with e_1 cannot proceed further because the edges in the first sub-loop are not available to it (by Property 4.5, since the first sub-loop was decomposed earlier) and the other edge in its path and e_1 are already shared by

the same complete sub-loop. Both chains do not end at **PE**, which means that the sub-loop remains incomplete. So we move the active sub-loop from the decomposed sub-loops list to **U**, because it has not been used.

4.6.1.2 One Chain Found

After processing the second sub-loop in Fig. 4.8a, we proceed to process the first one. Starting with its **PS**, three chains shown in Fig. 4.8c can be found based on the two rules: Chain 1 ($e_0-e_1-e_2-e_3$), Chain 2 (with e_4) and Chain 3 ($e_0-e_1-e_2-e_5$). Of the three chains, only Chain 1 ends at **PE**. The loop formed by the chain and the active sub-loop may be a face. For it to be, the following three conditions need to be met:

1. There is no chord within the loop.
2. No two edges in the loop intersect each other.
3. The loop does not contain a vertex that satisfies both the two following conditions:
 - 3.1 Its two adjacent edges in the loop have each been used by two other faces once;
 - 3.2 It connects to unprocessed sub-loops.

The first two conditions are well known [53], [45], [54] and [46]. Here, we only explain the third. The requirement 3.1 means that the vertex is used by some faces already formed. If the vertex satisfies both 3.1 and 3.2, the vertex must belong to known faces and unprocessed sub-loops. The missing chains for these unprocessed sub-loops must go through at least one of the edges adjacent to the vertex. Therefore, one or both of these adjacent edges will be required by unprocessed sub-loops to form real faces. If each of these adjacent edges has also been used once by a separate face, it cannot be used by the

active sub-loop, because one edge can only be shared by two real faces. Therefore, the loop cannot form a real face. We illustrate this with an example.

In Fig. 4.9, the edge chain in bold denotes the active sub-loop, the ones in dash are unprocessed sub-loops and the edges in gray have been used by two faces already formed and hence not available for the active sub-loop. Starting from ***PS*** of the active sub-loop, we can find two chains in the arrowed lines: Chain 1 (e_0 - e_1 - e_2 - e_3 - e_4) and Chain 2 (e_5 - e_6). All other candidates break either Rule F1 or F2. Both chains end at ***PE***, but only Chain 1 meets all the conditions. Chain 2 breaks the first condition since e_5 intersects an edge in the active sub-loop. It also does not satisfy the third condition because it contains one vertex (circled) with its adjacent edges, e_5 and e_6 , already used by other faces and the vertex connects to unprocessed sub-loops. As we can see from Fig. 4.9, both e_5 and e_6 are required by unprocessed sub-loops to form the two shaded faces. Therefore, only the loop formed by Chain 1 and the active sub-loop can be a real face.

To summarize, if a loop does not meet one of the first two conditions, it cannot be a real face. If it meets the first two conditions but not the third, it cannot be a real face but may be an internal face if ***I*** is not empty. If it meets all the three conditions, it is a real face.

Fig. 4.9. Another example for illustrating the face forming [55].

4.6.1.3 Multiple Chains Found

Fig. 4.10. An example which has multiple solutions.

There may be multiple chains that meet the three conditions. For the line drawing in Fig. 4.10a, the sub-loop marked in bold lines is the active one and the edges in dotted lines represent unprocessed sub-loops. Two chains can be found by starting from **PS** of the active sub-loop: Chain 1 (e_0 - e_1 - e_2) and Chain 2 (e_3 - e_4 - e_5). Both chains meet the three conditions, which means that both can potentially form a real face with the active sub-loop. If Chain 1 is selected, we get the loop shown in black in Fig. 4.10b. If Chain 2 is selected, we have the black loop in Fig. 4.10c. There is no way to know which one is correct at this juncture. So both loops are kept but flagged as alternatives. Further down the line, the two alternative gray loops would also be found. These two sets of loops share a virtual chord between **PS** and **PE**. Liu et al. [54] stated that two cycles cannot be real faces if they have the same virtual chord and enclose it. This rules out the combination in Fig. 4.10b, and leaves us with the solution in Fig. 4.10c.

It is also possible that both the combinations of the alternative loops are acceptable, leading to different face sets, as in the example in Fig. 4.11, first reported in [6]. Note that the primary difference between the two alternatives is in the top and back faces of the big

block, shown in bold in the drawing on the left. The other faces all have the same edges, but positioned differently when in 3D. In this case, our system would produce the alternatives and require the users to pick their preference.

Fig. 4.11. One line drawing with two different valid interpretations [6].

4.6.1.4 Face Confirmation

Most algorithms, such as those of Lipson and Shpitalni [53], Liu and Lee [45], Liu et al [54] and Leong et al [6], maintain faces found as potential faces, which are not yet real faces, until their co-existence with neighboring faces can be confirmed. In DFF, a face once formed must be a real face or an internal face. This can be proven fairly easily.

First we note that a sub-loop contains edges that must belong to the same face, real or internal. Its missing edges can only be found among faces already formed, as noted previously. In seeking the missing edges, the algorithm uses Rules F1 and F2 and the three conditions in Section 4.6.1.2 to ensure co-existence of the new face with the existing ones. Faces yet to be formed do not affect the validity of the current one. That means subsequent events cannot change the state of a face formed. Therefore a face once formed must remain a face.

This is a useful fact because it eliminates the checking for the validity of a face after it has been formed and thus saves on computations which other algorithms have to perform.

There are two exceptions, however. The first is the case of Section 4.6.1.3 when multiple face loops can be found for the same sub-loop. This is an ambiguous situation for which we store both faces found, and eliminate one of them later. Sorting out this ambiguity does not affect other faces. This is different from the uncertainty over the existence of every face until it has been checked against all its neighbours.

The other exception occurs in objects with holes, when it is not possible to distinguish between a hole loop and a face loop. A mistaken identity would lead to a wrong solution. This is a special case, and is discussed in Section 4.7.3.

4.6.2 The Full Face Forming Algorithm

Algorithm 4.2 is the full face forming algorithm. It basically goes through all the sub-loops in reverse to the order of their creation, starting from the last one, and forms a face with each. *F*, *IF* and *CL* are three lists used to store the faces, internal faces and candidate loops respectively. They are empty on entry.

Algorithm 4.2: face forming algorithm.

1)	<i>Store the last-loop as a face and place all its edges in CE</i>
2)	<i>Active sub-loop = the second last sub-loop formed</i>
3)	<i>Do</i>
4)	<i>If the active sub-loop is not complete, then</i>
5)	<i>Find PS and PE for the active sub-loop and update CE</i>
6)	<i>Use depth-first search to form loops by finding all possible chains connecting PS to PE using edges in CE based on Rules F1 & F2</i>
7)	<i>If the number of loops formed is 0, put the edges of the active sub-loop in U</i>
8)	<i>Else</i>
9)	<i>For each loop formed</i>
10)	<i>If it meets all three conditions in Section 4.5.1.2, save it in CL</i>
11)	<i>Else if it meets the first two but not the last, save it as an internal face in IF if I is not empty</i>
12)	<i>End For</i>
13)	<i>If the number of loops in CL is 0, put the edges of the active sub-loop in U</i>
14)	<i>Else if the number is 1</i>
15)	<i>Save the loop to F as a face</i>
16)	<i>Clear CL and update U.</i>
17)	<i>Else if the number is greater than 1</i>
18)	<i>Save all the loops to F and mark them as alternatives</i>
19)	<i>Clear CL and update U</i>
20)	<i>Update CE by removing edges in the chain and adding edges in active sub-loop</i>
21)	<i>Active sub-loop = the previous sub-loop</i>
22)	<i>While the first sub-loop has not been processed</i>
23)	<i>Resolve the alternative face loops if any</i>

4.6.3 Redundant Sub-loops

If a redundant sub-loop is produced during decomposition, then it would be visited during face formation. Invariably, there would not be a valid edge chain that can form a face loop with it, which is the situation described in Section 4.6.1.1. It is then placed in **U**, but unlike other edges in **U**, would be left unused upon the completion of face forming.

4.6.4 Complexity Analysis

All the steps in Algorithm 4.2 require constant or linear time with respect to the number of edges, except Step 6, where depth-first search is used to find the path of edges that

connects *PS* to *PE*. In theory, this depth-first search has exponential complexity over the number of candidate edges, e [45]. For an object with predominantly trihedral vertices, it would be $O(2^e)$. For an object with f faces, the complexity is $O(\sum_{i=1}^f 2^{e_i})$ where e_i is the number of available edges for the formation of the i^{th} face. It is not possible to provide a value for the summation because e_i varies for different faces.

In reality, the number of candidate edges is greatly reduced by reversing the face forming order from that of the decomposition and the removal of the edges that have already been shared by two faces. Hence in practice, the search paths are small in number and short in length. In our more than a hundred test cases, the largest number of search paths is 14 and the next largest is 10, both occurring only once. 59% of our examples have a maximum of only two paths, and 19% have three, 13% have four, and a small percentage for each of the rest up to 14. The length of each path is curtailed as well, given the limited number of candidate edges in *CE*. The longest path we have encountered in our tests consists of 48 edges, in an object where the largest face loop has 52 edges (Fig. 4.25m). It is typical for the longest path to have fewer edges than the largest face loop because we start from a sub-loop with two or more edges. When searching along an incorrect path, it is possible for the longest path to have more edges than the largest face. But among all the tests that we have done, this only occurred twice, the worst of which has twelve edges in its longest path and nine edges in the largest face. Practically, the number of search paths is small and can be considered to be bounded by a small constant, and the search length is in the order of the largest face, which is also a constant for a given object. This means that the depth first search is bounded by a constant, and hence in practical terms, the complexity of Algorithm 4.2 is $O(f)$.

4.6.5 Internal Face Identification

In the decomposition stage, we put the position number of a sub-loop in I once we detect the existence of an internal face in the remaining drawing. A sub-loop, s , whose position number is in I means, after decomposing s , there is the possibility of an internal face that connects the remaining drawing with the sub-loops decomposed before s . Take the line drawing in Fig. 4.12a as an example. Assume that Vertex i is the first start vertex, then, the position number, 5, of the fifth sub-loop is saved in I because the lowest degree of all the vertices in the remaining drawing is larger than 2 after its decomposition. Fig. 4.12 b shows the decomposed sub-loops (in dotted lines) and the remaining drawing. The loop in bold connects the remaining drawing to the decomposed sub-loops; it is an internal face. But in some cases, the loop that connects the remaining drawing to the decomposed sub-loops is not an internal face. For example, let the start vertex for the decomposition of the line drawing in Fig. 4.12c be Vertex j . The lowest degree of all the vertices in the remaining drawing is larger than 2 after decomposing the three sub-loops at j . Therefore, the number 3, for the third sub-loop, is recorded in I . But in this case, the loop in bold in Fig. 4.12d that connects the remaining drawing to the decomposed sub-loops is not an internal face as it cannot coexist with other real faces. However, in the face forming stage, we simply store it as a potential internal face. After forming all the faces, it will be filtered out in the subsequent internal face validation process.

After the decomposition, if I is not empty, internal faces potentially exist in the line drawing. In the face forming stage, we start from the last sub-loop to form faces using Algorithm 4.2. Before forming a face, we check the position number of the active sub-loop. If it is equal to the last number in I , there is potentially an internal face among the

faces. If there are multiple numbers in $\mathbf{I}$, multiple internal faces may exist. After finding an internal face, we delete the last number from $\mathbf{I}$. We analyze the relationship between each face and the unprocessed sub-loops, then, use the following property to judge whether a face is potentially an internal face.

Fig. 4.12. Examples to show the appearance of internal faces.

After the decomposition, if $\mathbf{I}$ is not empty, internal faces potentially exist in the line drawing. In the face forming stage, we start from the last sub-loop to form faces using Algorithm 4.2. Before forming a face, we check the position number of the active sub-loop. If it is equal to the last number in $\mathbf{I}$, there is potentially an internal face among the faces. If there are multiple numbers in $\mathbf{I}$, multiple internal faces may exist. After finding an internal face, we delete the last number from $\mathbf{I}$. We analyze the relationship between each face and the unprocessed sub-loops, then, use the following property to judge whether a face is potentially an internal face.

Property 4.7: *A face with three or more non-collinear vertices shared by unprocessed sub-loops and faces already formed is an internal face.*

Proof: Except for rare redundancies, unprocessed sub-loops will form faces. If three or more non-collinear vertices of a face, F , are shared by some unprocessed sub-loops and the same vertices are also shared by faces already formed, then, F connects the faces to be

formed with the unprocessed sub-loops and the faces already formed. It is therefore an internal face. ■

Fig. 4.13. Internal faces divide sub-loops into several groups.

Here, we use an example to demonstrate the identification of internal faces. We start from Vertex I to decompose the line drawing in Fig. 4.13a, which has three internal faces. The position numbers of the fifth, tenth and fifteenth sub-loops are saved in I during decomposition, because the lowest degree of all the vertices in the remaining drawing after decomposing them is larger than 2. These three numbers separate the decomposed sub-loops into four groups, as shown in Fig. 4.13b, c, d and e.

Start with the last sub-loop to form faces using Algorithm 4.2. After forming the first six faces shown in Fig. 4.14a, it is found that the position number of the sub-loop is equal to the last number in I . Hence, we need to find a face that satisfies Property 4.7. This face, the dotted loop in Fig. 4.14b, is the internal face we are seeking. Then, delete the last number from I and continue the face forming process. The other two internal faces found based on Property 4.7 are shown dotted in Fig. 4.14c and 4.14d; all the internal faces are saved in IF . These internal faces are subject to confirmation due to the situation depicted in Fig. 4.12d.

Fig. 4.14. An example to illustrate internal faces identification.

We note that if a drawing can be separated into two parts, A and B, by an internal face, then the faces in A are independent of B, and vice versa. For example, to identify the internal face (in dotted lines) in Fig. 4.14b, there is no need to use the unprocessed sub-loops (shown in Fig. 4.13b and c) decomposed before the internal face.

Internal faces require special treatment in some algorithms, especially those that prefer short loops as real faces ([6], [50], [55] and [57]). DFF delivers the internal faces and real faces separately. Existing methods may find the real faces, but they do not return the internal faces; this is true for [6], [45], [46], [53], [54], [55] and [58].

Although internal faces are not important in face finding, they are central to drawing separation as defined by Liu et al. [44], who proposed that complicated line drawings can be separated into multiple simpler ones across internal faces, and they provided several properties to identify internal faces with the assumption that all real faces are already known. If both the internal faces and the real faces are found correctly in the same face finding process, then, the separation becomes trivial.

4.6.6 Validation of Internal Faces

All the internal faces are stored in **IF**, but not all the faces in **IF** are real internal face.

There are two possibilities:

1. If each edge of an internal face is shared by two real faces and this internal face can coexist with all the real faces, it must be a real internal face.
2. If each edge of an internal face is shared by two real faces, but this internal face cannot coexist with one or more real faces, it cannot be a real internal face.

The loop in bold in Fig. 4.12b belongs to the first situation, it is an internal face, whereas the loop in bold in Fig. 4.12d belongs to the second; it is not an internal face.

4.6.7 Correctness of the Results

Leong et al [13] used two rules to validate the correctness of a set of faces for manifolds.

Rule M1: Each edge must be shared by exactly two faces.

Rule M2: The resulting faces must satisfy the Euler-Poincare equation.

We incorporate the same validation procedure in DFF. Except for objects with a certain type of holes, this validation is not necessary theoretically because the faces found by our algorithm are always correct, as shown in Section 6.1.4. However, the check is useful in guarding against coding errors.

The type of holes alluded to above are those sharing edges with internal faces. Errors may arise because hole loops may be mistaken as face loops, as they have the same topology. The validation is necessary then. Section 7.3 deals with this issue in full.

4.6.8 Summary

The face forming algorithm for general objects (which may contain internal faces) is summarized in Algorithm 4.3.

Algorithm 4.3: the face forming algorithm for general objects.

- 1) *Check the number of items, n , in $\mathbf{I}$.*
- 2) *If $n=0$ (no internal face)*
- 3) *Run Algorithm 4.2.*
- 4) *Else*
- 5) *Run Algorithm 4.2 until the position number of the active sub-loop equals the last number in $\mathbf{I}$.*
- 6) *Identify the internal face out of the faces based on Property 4.7 and save it in $\mathbf{IF}$.*
- 7) *Delete the last number from $\mathbf{I}$.*
- 8) *If the number of unprocessed sub-loops $\neq 0$, go to step 5.*
- 9) *Validate the internal faces.*
- 10) *End.*

4.7 Discussions

There are four special and interesting issues which require further discussions.

4.7.1 Non-manifolds

It would not have escaped notice that after the first decomposition of a manifold, the remaining drawing is usually that of a non-manifold. But the subsequent decompositions and face forming remain unchanged. This means that the algorithm applies to both manifolds and non-manifolds alike.

However, there are four issues that require further attention.

4.7.1.1 The Last Loops Self-intersects

For non-manifolds, the last sub-loop may not form a face because it may have self-intersection. For example, in Fig. 4.15, the loop *abcd*a may be the last sub-loop. It self-intersects and therefore cannot be a face. This means that the object must be a non-manifold. Hence, instead of taking this last loop as a face loop, as is done for manifolds,

we simply place the whole loop in U , so that it does not form a face loop and yet its edges are available for use by subsequent sub-loops to form faces.

Fig. 4.15. An object with a loop that has self-intersection.

4.7.1.2 Edges Shared by One or More Real Faces

For manifolds, the number of real faces that can share an edge is two and only two. But for non-manifolds, it can be any number starting from one; Fig. 4.16 gives an example. This means that Rule F2 is not applicable for non-manifolds. Instead we adopt a rule first used by Shpitalni and Lipson [53] and subsequently adopted by Leong et al [6]:

Rule N1: An edge must be shared by at most $d-1$ faces, where d is the smaller of the degrees of the two end vertices of the edge.

Fig. 4.16. A non-manifold containing an edge shared by more than two faces.

4.7.1.3 Internal Faces and Non-manifolds

Internal faces also require special attention in non-manifolds. If an internal face can coexist with all the real faces, but not all its edges are shared by two or more real faces, it must be a real face instead.

For example, in Fig. 4.17a, we can find seven real faces and one internal face (in dotted lines). But two of the edges (in bold dotted lines) are only shared by one real face, so it is not an internal face but a real face instead. There are therefore eight real faces. In Fig. 4.17b, we can obtain 20 real faces and four internal faces, marked in dotted loops. But every internal face contains some edges (in bold dotted lines) shared by only one real face. Therefore, all of the internal faces are real faces, making a total of 24 real faces.

Fig. 4.17. Real faces that were regarded as internal faces in the face forming stage.

4.7.1.4 Validation

Neither of the rules for the validation of manifolds holds for non-manifolds. Instead, we use Rule N1 stated in Section 4.7.1.2.

Fig. 4.18. A non-manifold which is treated as a manifold.

Because the rules are different for manifolds and non-manifolds, it is necessary to determine whether a drawing depicts a manifold or a non-manifold. Non-manifolds can be divided into two types: (1) simple non-manifolds and (2) combination of manifolds,

simple non-manifolds and dangling edges [6]. The lowest degree of a vertex of a manifold is 3, hence an object with vertices of degree less than 3 must be a non-manifold. Some non-manifolds that belong to the second type can be partitioned into manifolds along non-manifold vertices or edges, such as the objects shown in Fig. 1. But there exist non-manifolds with all their vertices of degree above 2 and yet cannot be partitioned; the drawings in Fig. 4.15 and 4.18 are examples. The object in Fig. 4.18 is a non-manifold because no face set we can form with its nine vertices and 20 edges can satisfy the Euler-Poincare equation. But the degrees of its vertices are all larger than 3 and it cannot be partitioned into simpler manifolds or non-manifolds. As yet, we do not have a way to determine that the drawing represents a non-manifold without face information. In DFF, it is taken as a manifold. Therefore, even if the correct face set is found for a non-manifold, it would be validated as a manifold and therefore fail. Our solution to such a situation is: if we cannot find a face set that satisfies rules M1 and M2 for an object, then, we check whether it satisfies N1. If it does, provide the user with the face set for a non-manifold. Otherwise, inform the user that no valid face set can be found.

4.7.2 Drawings that Contain Multiple Internal Face Sets

The second issue is multiple internal faces in a drawing. Depending on the start vertex chosen, different internal faces can be discovered. For instance, there exist three possible internal face sets shown in bold in Fig. 4.19b, c, and d for the drawing of Fig. 4.19a. One set can be found in one run of the algorithm; Vertex i , j or k as the start vertex would result in the internal face sets shown in Fig. 4.19b, c and d respectively.

Fig. 4.19. A drawing and its three possible internal face sets (in bold).

4.7.3 Drawings that Contains Holes

Finding the faces of objects with holes can be tricky. At this juncture, we are not able to determine the presence of through holes in a drawing automatically, so a user needs to indicate their number in an object. In common with [6], a hole loop in a face that is not directly connected to the face is not treated as a hole loop because, while one may perceive the face and the hole to be in the same plane, the drawing does not provide the information to ascertain that. Hence in Fig. 4.20, the outer loop in black and the inner loop in gray at the bottom are not considered to be in the same face; the inner loop is treated as a face loop and the genus of the object is 0.

Fig. 4.20. The two loops at the bottom are not treated as in the same plane.

In some special situations, DFF may not find the faces of objects with holes correctly as a hole loop may be treated as a face instead. This occurs when hole loops adjoin internal

faces; see Fig. 4.21 for an example. As explained in Section 5.1, decomposed sub-loops can be divided into different groups separated by internal faces. If there exist hole loops adjoining internal faces in an object, sub-loops of holes may be formed instead of sub-loops of faces, leading to incorrect results. This can be corrected by changing the order of the sub-loop decomposition. We will illustrate this phenomenon shortly. Simple hole loops, i.e. holes not adjoining internal faces, such as the one that appears in the top of the object in Fig 4.20, poses no problem.

During decomposition, there may be multiple vertices with the lowest degree in the current drawing. We can choose any one of these vertices as the start vertex for further decomposition, and a different vertex would lead to a different decomposition order. For objects without holes, this would not affect the correctness of the result. But for objects with holes, a different order could lead to different results. Take Fig. 4.21a again, with Vertex i as the first start vertex, after decomposing the fifth sub-loop, the remaining drawing has vertices all of degree larger than 2, suggesting the existence of internal faces, as Fig. 21c shows. Recall that sub-loops are in groups separated by internal faces (see Section 5.1). So, the first five sub-loops are in the same group shown in Fig. 4.21b. There are 12 vertices with the lowest degree in the remaining drawing, marked using two different symbols, circle and dot. Decomposing from a vertex in circle, say Vertex j , would produce the group of sub-loops shown in Fig. 4.21d, with the remaining drawing shown in Fig. 4.21e. Decomposing from a vertex in dot, say Vertex k , would produce the group of sub-loops shown in Fig. 4.21f, with the remaining drawing in Fig. 4.21g. Both the remaining drawings in Fig. 4.21e and 4.21g lead to simple objects without hole, and the face forming process can form their faces. But, of course, Fig. 4.21g is actually the

hole, and forming faces with it would lead to the wrong result. This mistake can be rectified by choosing a vertex in circle instead of a vertex in dot as the start vertex.

Fig. 4.21. Two possible decomposition results for a drawing.

The sub-loops containing these vertices are in groups, demarcated by the positions of the internal faces held in I . Changing the order of the group formation allows the selection of a vertex from a different group as the start vertex, and avoids faces from being formed over holes. If the drawing contains n groups of sub-loops, the maximum number of ways of ordering the groups is the permutation of the groups, P_n^n , which is $n!$. That is to say, to obtain the correct faces of a drawing with such holes, we may need to perform the decomposition and the face forming processes $n!$ times. Fortunately, there are very few objects with a hole adjoining internal faces.

4.7.4 Drawings with Multiple Interpretations

Fig. 4.22. Objects with multiple valid interpretations.

Some drawings have multiple valid interpretations. For example, the drawing in Fig. 4.22a has three valid interpretations, shown in Fig. 4.22b-d (loops in bold are hole loops). This example is well known; most face finding algorithms are able to obtain the three valid results. The drawing in Fig. 4.22e has two valid interpretations shown in Fig. 4.22f-g. For this case, algorithms that prefer short loops as real faces, such as those of [50], [57], [55] and [6], will miss the second interpretation, because they prefer the loop *abcjiha* as a face, while the second interpretation must take the face *abcdefgha*. DFF can find all the valid interpretations, because it does not prefer short faces. But one run of the algorithm can only produce one valid interpretation. The others can be obtained in a new run using a different start vertex during decomposition.

4.8 Results

We have implemented the algorithm and tested it using many drawings, including all those in the published literature. For conciseness, we will show only a proportion of the

results. We also compare our results with those of Varley and Company [55] and Leong et al [6].

4.8.1 Test Objects and Run Times

Fig. 4.23 show some test drawings that contain no internal faces. The computation time and the number of real faces found are listed below each example. The drawings in Fig. 4.23a–4.23d are from [55] and the ones in Fig. 4.23e–4.23h are from [12]; the rest are our own. Some objects are purely trihedral while some have vertices of degree greater than 3. The objects in Fig. 4.23c and Fig. 4.23i have one blind hole while that in Fig. 4.23j has three through holes. The objects in Fig. 4.23k- 4.23l are non-manifolds.

Fig. 4.24 shows some test objects with internal faces. The examples in Fig. 4.24a–4.24d are from [55] and the ones in Fig. 4.24d and 4.24e are from [44] and [45] respectively; the others are our own. Figs. 4.24e and 4.24l are objects with holes adjoining internal faces; the latter takes a lot longer for the number of faces it has because the algorithm has to be repeated after swapping sub-loop groups to find the correct faces. Fig. 4.24g is a non-manifold. Fig. 4.24k is an object with no trihedral vertex.

We use three sets of objects to determine the actual computation time. The three sets of objects are buildings in Fig. 4.25a, desks in Fig. 4.25b and stairs in Fig. 4.25c. These objects are all from [55]. The relation between the computation time and the number of edges are plotted in a chart in Fig. 4.26. It is observed that the computation time increases linearly with increasing of the number of edges.

(a) 0.70ms, 22	(b) 0.48ms, 19	(c) 2.60ms, 32	(d) 2.97ms, 33
(e) 0.32ms, 16	(f) 0.92ms, 22	(g) 0.39ms, 18	(h) 0.85ms, 22
(i) 0.30ms, 15	(j) 3.47ms, 18	(k) 0.44ms, 10	(l) 0.57ms, 12

Fig. 4.23. Test objects without internal faces. The numbers are the DFF run time in milliseconds and the number of faces.

4.8.2 Comparison with Existing Method

We now compare the performance of DFF with that of others. There are numerous face finding algorithms and it is impractical for us to compare with all of them. We only make comparison with the two most recent works, that of Varley and Company [55] and the double breadth-first search method by Leong et al [6]. For brevity, we refer to the former as V&C while the latter as DBFS.

Both V&C and DBFS give preference to creating smaller faces, such as triangles and quadrilaterals, which leads to complications when these faces are not the real faces. DFF has no such preference.

Fig. 4. 24. Test objects with internal faces.

(a) Buildings.

(b) Desks.

(c) Stairs.

Fig. 4.25. Buildings, desks and stairs.

4.8.2.1 Range of Objects

V&C only handles manifolds; it does not handle non-manifolds and objects that do not contain trihedral vertex. DBFS handles all manifolds and simple non-manifolds. It does not deal with non-manifolds that are not simple non-manifolds and non-manifolds of the type shown in Fig. 4.18, which cannot be distinguished from a manifold without the faces. DFF handles all manifolds and non-manifolds, though non-manifolds of the forms of Fig 4.1 need to be partitioned into the constituting components first. If a drawing has multiple

valid interpretations, DFF can find all of them, but V&C and DBFS may not. The object in Fig. 4.22e is such an example.

Fig. 4.26. The relation between time and the number of edges in buildings, desks and stairs.

4.8.2.2 Efficiency

Theoretically, the worst case time complexity for V&C is $O(e^5)$, where e is the number of edges. In practice, the complexity is between $O(e^2)$ and $O(e^3)$ [55].

For DBFS, if the faces can be found correctly the first time, the time complexity is $O(e^2)$, where n is the number of edges. Otherwise, in the worst case, the time complexity is $O(e(D)^{2e})$, where D is one less than the maximum degree of the vertices in the drawing. But experimental results in [6] show that the practical complexity is linear.

In DFF, sub-loop decomposition is an operation local to a vertex and its complexity is $O(v)$, where v is the number of vertices. For face forming, in practical terms, the complexity is $O(f)$, where f is the number of faces, as explained in Section 6.4. Most drawings require only one run of the decomposition and the face finding algorithms. So

the overall complexity is $O(v+f)$. In the rare situation of a drawing containing holes adjoining internal faces, then the decomposition and face forming may have to be repeated $n!$ times where n is the number of sub-loop groups in the drawing. Hence in the worst case, the complexity is $O((v+f)*n!)$.

The V&C algorithm only runs once for a drawing. For those objects that cannot be handled correctly, it does not go back to try to find the correct answer. DBFS checks for the correctness of the results, and may need to go through up to five stages with different constraints to find the correct answer, but a large majority of the solutions can be found in the efficient first stage. The DFF algorithm guarantees that the faces of the object found are always correct, so there is no need to check for correctness except for the case of a hole adjoining internal faces.

Table 4.1 Run time comparison between DBFS and DFF, time in milliseconds.

Object	Number of faces	DBFS time	DFF time
(a)	18	102.9	0.2
(b)	21	628.8	0.3
(c)	12	53.0	0.3
(d)	15	16.4	0.1
(e)	20	6.4	0.3
(f)	10	124.0	0.1
(g)	42	18.3	4.8

Object	Number of faces	DBFS time	DFF time
(h)	26	75.1	0.9
(i)	28	23.4	1.4
(j)	36	31.2	3.5
(k)	31	15.6	2.8
(l)	42	136.9	12.5
(m)	64	217.7	32.5
(n)	73	306.0	21.3

Fig. 4.27 shows some objects we use to compare DFF and DBFS. The quantitative results are given in Table 4.1. Such a direct comparison is possible because DBFS was also

developed by the authors. No meaningful time comparison with V&C can be made since we do not have their implementation.

The drawings in Fig. 4.27a-e and n are from [6], the drawings in Fig. 4.27g-m are from [55] and the one in Fig. 4.27f is new. For some of these drawings DBFS needs to repeat the algorithm several times to obtain the correct answer.

Table 4.1 shows that DFF always out performs DBFS, often tens of times faster.

Fig. 4.27. Examples used to compare the efficiency between DFF and DBFS.

4.9 Chapter Summary

The novel algorithm, DFF, presented in this paper is totally different from all the methods that have been reported in the literature for finding faces in line drawings. At a high level, DFF's decomposition and then face forming is intuitively simple. Its strength is derived

from the decomposition of the drawing into sub-loops that must belong to one face only. This decomposition is local to a vertex, regardless of the complexity of the drawing as a whole. Always starting a decomposition at a vertex of the lowest degree minimizes the number of redundant sub-loops. Reversing the order of face forming from the order of sub-loop decomposition reduces greatly the search space for candidate edges to complete a sub-loop. These two mechanisms are fundamental to the efficiency of the algorithm, which is typically tens of times faster than DBFS, and we believe even faster still than other algorithms.

Existing algorithms deliver only the real faces, whereas DFF delivers both the real faces and the internal faces. In common with most existing algorithms, DFF can handle both manifolds and non-manifolds using the same process, except for some special cases peculiar to non-manifolds. Objects with holes present no complications except for holes that adjoin internal faces, which is overcome by working through the permutation of sub-loop groupings partitioned by internal faces. This permutation gives rise to a factorial complexity over the number of groups. In our view, this is the weakest link in the whole algorithm. But the issue is encountered rarely. Still, we are working to seek an alternative, particularly in the ability to identify holes directly.

CHAPTER 5

The Cubic Corner Method for 3D Reconstruction

5.1 Introduction

This chapter presents a direct method to recover the geometry of the 3D polyhedron depicted in a single parallel or perspective projection. For parallel projection, it uses two sets of information, the list of faces in the object, obtained automatically from the drawing, and a user-identified cubic corner, to compute for the coordinates of the vertices in the drawing and thus establish the 3D geometry of the whole polyhedron. As to perspective projection, before proceeding to compute the coordinates of the vertices in the drawing, focal length need to be found based on vanishing point information to establish the linear relationship between a vertex on the image plane and its corresponding point on the object. The algorithm exploits the topological structure of the polyhedron, implicit in the connectivities between the faces, resulting in a complexity linear in the number of faces. The method is extended to objects with no cubic corners as well. The algorithm works well for recovering objects from accurate line drawings (parallel or perspective projection), producing accurate 3D objects.

5.2 Cubic Corner

Many man-made objects, especially mechanical engineering and architectural objects, contain cubic corners, where the three, and only three, edges meeting at a corner are mutually perpendicular.

5.2.1 Cubic Corner in Parallel Projection

Perkins coined the term cubic corner and established the relationship between a cubic corner in 3D and its 2D projection [59]. Fig. 5.1 shows a cubic corner and its parallel projection. Perkins' formulae enable the derivation of the 3D coordinates of the three end points of the three edges at the cubic corner, in relation to the coordinates of the cubic corner itself. Assuming that the direction of the projection is Z, then, the x, y coordinates of the end points of the three edges in the projection are the same as the x, y coordinates of the corresponding points in 3D. The unknowns are the z -coordinates of the three end points in 3D.

Fig. 5.1. A cubic corner and its 2D projection.

Let P, Q and R be unit vectors in the directions of the three lines in the projection, and the angles between them be a, b and c , as denoted in Fig. 5.1. Assume that the z -coordinate of the cubic corner itself is 0, and p, q and r are the z -coordinates of the other end points of the three corresponding 3D edges to P, Q and R , Perkins showed that [59]

$$p = \pm \sqrt{-\frac{(\cos a)(\cos c)}{(\cos b)}}, \quad q = \pm \sqrt{-\frac{(\cos a)(\cos b)}{(\cos c)}}, \quad r = \pm \sqrt{-\frac{(\cos c)(\cos b)}{(\cos a)}} \quad (5.1)$$

where $\cos a, \cos b, \cos c \neq 0$. Coupled with the (x, y) coordinates of the end points from their 2D projections, the p, q and r values completely define the three end points connected to the cubic corner.

However, there are two values each for p, q and r , which mean there are totally eight possible combinations for the three end points. According to the Necker's cube illusion, there are two, and only two, correct combinations, each of which must form an orthogonal set. To obtain the two, we choose one vector, say Q , which has two z values, q and $-q$. We first let the z value of Q be q , then the z values of the other two end points can be uniquely determined according to the orthogonal relationship among the three edges at the cubic corner. Then, we set the z value of Q to $-q$ to get the other combination.

Several other researchers have studied or used cubic corners in their work. Cooper [10] and Varley et al. [60] examined the use of cubic corner in conjunction with line-labeling. Andre et. al. [61], [62] exploited cubic corners in an attempt to reconstruct freeform objects from line drawings, with the postulation that crossing lines on a surface are often perceived as perpendicular to each other in 3D.

Suh [63] exploited cubic corners in his attempt to reconstruct extruded objects, where the direction of extrusion is perpendicular to the plane containing the extrusion profile. But the result here is applicable to a very limited domain of extruded objects. To increase the geometric coverage, Suh used Boolean operations to create objects containing more than one extruded part.

5.2.2 Cubic Corner in Perspective Projection

Upon specifying a cubic corner in a perspective projection, we then determine the coordinates of the four vertices at the cubic corner. Fig. 5.2 shows a cubic corner and its perspective projection on the image plane.

The points P, Q, R and C form a cubic corner in 3D space with C as the chosen corner, and their perspective projections on the image plane are p , q , r and c respectively. The coordinates of the points p , q , r and c can be read from the line drawing. To obtain the coordinates of points P, Q, R and C, we use the relationship between a point on the 3D object and its corresponding vertex on the image plane

$$X = Z \times x / f \quad (5.2a)$$

$$Y = Z \times y / f \quad (5.2b)$$

where X , Y and Z are the coordinates of the point in 3D space; x and y are its coordinates on the image plane; f is the focal length which is unknown so far.

Fig. 5.2. A cubic corner and its projection on the image plane.

The focal length determines the appearance of the picture in the camera. To recover the object in the picture, it is therefore necessary to determine the focal length, the computation of which relies on the vanishing points.

5.2.2.1 Vanishing Points

Determining the vanishing points in a drawing is a well-studied problem; see, for example, [64] and [65]. In this chapter, we employ the method reported in [65]. This method has two major steps: accumulation and search. In the accumulation step, potential vanishing points are found from the pairwise intersection of all the input lines. For every intersection point, a count is kept on the number of line segments within a given distance to it. The larger this count, the more likely the point would be a vanishing point. The task of the search step is to determine the vanishing points which correspond to three mutually orthogonal directions of the scene from the pool of potential vanishing points. Two criteria are involved: orthogonality and camera. Orthogonality requires that the three directions from the view point to the three vanishing points are mutually orthogonal; camera requires that the principal point, which is the projection of the view point on the image plane, and the focal length to be inside a certain range. The potential vanishing point with the largest number of votes is chosen as one of the three vanishing points first. Then, the remaining potential vanishing points are picked in pairs and checked with the first point to find the three that best satisfy the orthogonality and camera criteria.

5.2.2.2 Focal Length

With the vanishing points known, the problem of getting the focal length has been well studied; see, for example, [66] and [67].

For a drawing in two-point perspective, let $V_1(x_1, y_1)$ and $V_2(x_2, y_2)$ be the two vanishing points. The focal length can be computed using the following formula [66],

$$f = \sqrt{-(x_1 \times x_2 + y_1 \times y_2)} \quad (5.3)$$

For three-point perspective, let $V_1(x_1, y_1)$, $V_2(x_2, y_2)$ and $V_3(x_3, y_3)$ be the three vanishing points in the drawing plane. Let S be the view point. Let S' be the orthocenter of $\Delta V_1V_2V_3$; that is, the lines V_1V_1' , V_2V_2' and V_3V_3' , which are perpendicular to the three sides V_2V_3 , V_3V_1 , and V_1V_2 respectively, meet at S' (see Fig. 5.3). The optical axis, going from S to S' , is normal to the drawing plane and its length is the focal length [67]. It can also be shown that SV_1 and SV_1' are perpendicular to each other. Likewise, SV_2 is perpendicular to SV_2' and SV_3 is perpendicular to SV_3' .

Fig. 5.3. Geometric construction of the focal length with the triangle of vanishing points.

Hence $\Delta SS'V_1$ and $\Delta SS'V_1'$ are similar, and the focal length, f , can be obtained thus:

$$f^2 = |SS'|^2 = |V_1S'| \times |V_1'S'| \quad (5.4a)$$

Note that this formula depends purely on the geometry of $\Delta V_1V_2V_3$, without recourse to the position of the view point S which, until the finding of the focal length, is unknown.

Similarly, it can be established that

$$f^2 = |\mathbf{S}\mathbf{S}'|^2 = |\mathbf{V}_2\mathbf{S}'| \times |\mathbf{V}_2'\mathbf{S}'| = |\mathbf{V}_3\mathbf{S}'| \times |\mathbf{V}_3'\mathbf{S}'| \quad (5.4b)$$

5.2.2.3 Z Coordinate's Derivation for Cubic Corner in Perspective Projection

With the focal length, the coordinate of a point in 3D can be derived from its corresponding vertex on the image plane with only one unknown z based on Eq. (5.2a) and (5.2b). Let the coordinates of the points p , q , r and c be $p(x_p, y_p)$, $q(x_q, y_q)$, $r(x_r, y_r)$ and $c(x, y)$. The corresponding end points in 3D can be obtained in terms of the z coordinates:

$$\mathbf{P} = (Z_p \times x_p / f, Z_p \times y_p / f, Z_p) \quad (5.5a)$$

$$\mathbf{Q} = (Z_q \times x_q / f, Z_q \times y_q / f, Z_q) \quad (5.5b)$$

$$\mathbf{R} = (Z_r \times x_r / f, Z_r \times y_r / f, Z_r) \quad (5.5c)$$

The z coordinate of the cubic corner C can be an arbitrary value. A different value will merely scale the object proportionally. Here, we assume that C is on the image plane, so its z coordinate is equal to the focal length f . So C 's coordinates are (x, y, f) . We know that the three sides, PC , QC and RC , are mutually perpendicular in 3D space. Hence, taking dot products, we have

$$(\mathbf{P}-\mathbf{C}) \cdot (\mathbf{Q}-\mathbf{C}) = 0, (\mathbf{Q}-\mathbf{C}) \cdot (\mathbf{R}-\mathbf{C}) = 0, (\mathbf{R}-\mathbf{C}) \cdot (\mathbf{P}-\mathbf{C}) = 0 \quad (5.6)$$

Substituting the coordinates of P , Q , R and C into Eq. (5.6), we get

$$\left(\frac{x_p \times Z_p}{f} - x\right)\left(\frac{x_q \times Z_q}{f} - x\right) + \left(\frac{y_p \times Z_p}{f} - y\right)\left(\frac{y_q \times Z_q}{f} - y\right) + (Z_p - f)(Z_q - f) = 0 \quad (5.7a)$$

$$(\frac{x_p \times Z_p}{f} - x)(\frac{x_r \times Z_r}{f} - x) + (\frac{y_p \times Z_p}{f} - y)(\frac{y_r \times Z_r}{f} - y) + (Z_p - f)(Z_r - f) = 0 \quad (5.7b)$$

$$(\frac{x_r \times Z_r}{f} - x)(\frac{x_q \times Z_q}{f} - x) + (\frac{y_r \times Z_r}{f} - y)(\frac{y_q \times Z_q}{f} - y) + (Z_r - f)(Z_q - f) = 0 \quad (5.7c)$$

We have three non-linear equations with only three unknowns, Z_p , Z_q and Z_r . This system of equations can be solvable in closed form, producing two values for the z coordinates of each end point. Therefore, there are totally eight possible combinations for the three end points. The same situation occurs for parallel projection, where there are only two correct combinations, which correspond to the two solutions in the Necker's cube illusion [68]. But under perspective projection, only one solution is correct. This cannot be resolved using the cubic corner alone, but can be done with the aid of the vanishing points in the drawing which contains the cubic corner. If two points lie on a line that crosses one of the three vanishing points, the point closer to the vanishing point should have a larger z coordinate. That is because in perspective view, objects closer to the vanishing points are further away and appear smaller. Once one of the three end points is known, the other two can be uniquely determined using orthogonal relationships. Hence, our task is to select the correct value for any one of the three end points. Given the cubic corner, we choose one of its three end points, say p .

The search for the vanishing points using Rother's algorithm (see Section 5.2.2.1) establishes three sets of edges in the drawing that intersect at the vanishing points. Within each set, the edges are parallel in 3D, but the sets are mutually orthogonal. Many of the edges would have originated from orthogonal vertices in the drawing, providing a simple and direct means for automatically identifying a cubic corner, which is a vertex shared by

three edges, one from each of the three sets. By using one of these as the chosen cubic corner, we guarantee that every of the three crosses a vanishing point. Hence, the cubic corner, p , and one of the vanishing points are in a straight line. If p is closer to the vanishing point than the cubic corner along the line, then the z coordinate of p must be larger than that of the cubic corner; otherwise, the z coordinate of p is less than that of the cubic corner.

5.3 Reconstruction of 3D Objects

3D reconstruction requires face information. The problem of automatic face finding has been addressed in Chapter 4.

A line drawing may have more than one valid interpretation as a 3D object. The different interpretations are manifested in the different valid sets of faces possible, as the same face set would always define the same object. It is therefore the “duty” of the face-finding algorithm to produce the desired set of faces. It could, for example, present the different valid choices to the user who then has to choose the desired one. The 3D reconstruction algorithm we present here simply makes use of the face set it receives.

With the faces of the object identified, what remains now is to find the 3D coordinates of its vertices.

If a cubic corner exists in the object, Eq. (5.1) or (5.7) can be used to obtain the (x, y, z) coordinates of the end points of the three edges connected to the cubic corner. Each pair of edges contains three vertices, which uniquely determine the plane of the face bounded by the edge pair. The equation of a plane can be written as

$$Ax + By + Cz + D = 0 \quad (5.8)$$

where A , B , C and D are the coefficients which can be found from the three known vertices.

In parallel projection, the (x, y) coordinates of the vertices in the boundary of a face can be read from the drawing directly. In perspective projection, the (x, y) coordinates are $(x' * z/f, y' * z/f)$, where (x', y') are their corresponding coordinates on the image plane. Therefore their z -coordinates, being the only unknown, can be found from Eq. (5.7). Hence, the vertices in the three faces meeting at the cubic corner can all be found. The problem that remains is to determine the 3D coordinates of all the remaining vertices in the drawing.

Take the drawing in Fig. 5.4 as an example. Assume that Vertex a is the chosen cubic corner. Hence its coordinates are known. Eq. (5.1) enables the determination of the 3D coordinates of vertices b , d and f if the drawing in Fig. 5.4 is a parallel projection, otherwise Eq. (5.7) can be applied. These establish the planes of faces $abcd$, $adef$ and $abgf$, which then lead to the 3D coordinates of vertices c , e and g respectively. Only vertices h , i and j remain unknown.

Fig. 5.4. An object with seven faces. The chosen cubic corner is marked with a dot.

In an object with connected faces, the initial three known faces share their vertices with their adjacent faces. So each of these adjacent faces has some known vertices. For an adjacent face with three or more known non-collinear vertices, its plane and hence the rest of its vertices can be found. It is therefore a straightforward matter of systematically visiting every face, finding its plane equation from the coordinates of its known vertices, and then finding the coordinates of the vertices still unknown, until all the vertices are found.

In the case of the object in Fig. 5.4, the known vertices b , c and g determine the plane of Face $bcjhgb$, which then leads to the coordinates of vertices h and j . Similarly, the plane of Face $cdeijc$ can be found from the known vertices c , d and e , which further determines the remaining vertex i , thus completing the reconstruction of all the vertices, and therefore the 3D geometry of the object.

Note that there are multiple routes to finding the coordinates of vertices h , i and j . Assuming that the drawing is accurate, all the routes would lead to the same values for the unknown vertices. However, if the drawing is inaccurate, then the different routes will produce different results. This might seem a problem but is, in fact, something we can exploit to advantage. We will study this issue further in Section 5.4.1. First we will show that it is possible to completely recover a cubic graph, which is a graph with all its vertices trihedral.

Theorem 5.1: For an object whose vertices and edges form a cubic graph, the planes of all its faces, and hence the coordinates of all its vertices, can always be determined starting from any given corner.

Proof: This proof consists of two steps. The first shows that it is possible to recover all the faces adjacent to a known face. The second shows that, because of the connectivities of the faces in a cubic graph, it is possible to visit and recover all the faces.

Two faces are adjacent when they share an edge. For a cubic graph, at each end point of this common edge are two and only two other adjacent edges, each belonging to one of the two faces. If the planes containing the two faces are known, then all their vertices are known. Then these two edges, containing three known vertices, define the plane of the third face that contains them. In Fig. 5.5, after the cubic corner computations, Face *abgfa* is known and has *adefa* as its known neighbour. These two faces have adjacent edges *fg* and *ef*, which contain the three vertices *e*, *f* and *g*. This means Face *efghie*, which contains the two edges, is known. Systematically going round the edges of Face *abgfa*, we can find all its adjacent faces.

To show that it is possible to traverse all the faces in the object in general, let Face 1 be one known face, with ordered edges 1, 2, ..., *n*, in its bounding edge loop, where *n* is the number of edges. Let Face 2₁ be its known neighbour, sharing Edge 1 between them. (See Fig. 5.5.) This means that the edges incident to the end vertices of Edge 1 are all known. At one end, they are Edge 2 in Face 1 and Edge *e* in Face 2₁. This means that Face 2₂, which shares Edge 2 with Face 1 and Edge *e* with Face 2₁ can be found. Taking this process progressively through the edges of Face 1, we can find all its *n* adjacent faces. Furthermore, we can find all the faces adjacent to Face 2₁, Face 2₂, etc. using the same process. The degree three of the vertices ensures that every unknown face can have two known faces adjacent to it and thus can be found. Given that the faces are all connected,

this means that all the faces can be found through a systematic traversal. This completes the proof. ■

For an object that is not a cubic graph, it may not be possible to visit all the faces, which means that it may not be possible to recover the complete object. This issue is further elaborated in Section 5.4.2.

Fig. 5.5. Faces adjacent to a given face.

Assuming that full face traversal is possible, the algorithm for the reconstruction of 3D polyhedral objects from an accurate line drawing can be summarized thus:

Algorithm 5.1: reconstruction of polyhedral objects from an accurate line drawing.

- 1) *Find the faces of the object in the line drawing, represented by their lists of vertices*
- 2) *Identify a cubic corner and assume its z -coordinate to be 0*
- 3) *Compute for the z -coordinates of the end points of the edges at the cubic corner*
- 4) *Starting from the faces that meet at the cubic corner, for every face in the object*
- 5) *Find the equation of the plane of the face from three known vertices*
- 6) *Find the z -coordinates of the unknown vertices in the face*
- 7) *Until all the vertices in the drawing have been processed*

Given the faces, the worst case complexity of the 3D reconstruction algorithm is linear in the number of faces, since every face needs to be visited at most once. In fact, often not every face needs to be visited before all the vertices are found. For example, there is no need to visit the faces $efghie$ and $hijh$ in Fig. 5.4 because all their vertices have been

determined prior to the visit. Therefore on average the complexity is less than linear in the number of faces.

5.4 Discussion

The reconstruction of the 3D object from an accurate parallel or perspective projection is straightforward, provided a cubic corner exists. But this is not really the final destination in our study, because accurate drawings are not the norm in some applications, such as conceptual design, and not all objects contain cubic corners, even though the majority of architectural and mechanical objects do. It is more interesting, useful and technically a lot more challenging to produce the 3D objects depicted in sketches, which are normally inaccurate and often incomplete.

Sketching is still a medium of choice for many designers during initial conceptual design for its convenience and freedom from the constraints of user-interface gadgets. It would be very useful, as a visualization aid and more, if the designer can be presented with the 3D object from a sketch, inaccurate and incomplete it may be.

All issues we discuss in this section are applicable to both parallel and perspective projections; we illustrate using only parallel projection.

5.4.1 Multiple Paths to a Vertex

For an accurate drawing, the multiple paths that lead to the computation of the coordinates of a vertex, alluded to earlier, would produce the same result. However, for inaccurate drawings, the different paths would produce different results. It is difficult to say which of these results is the best, or whether anyone of them is good enough. One

possible resolution to the quandary is to take the average of the results from different paths and return it as the final outcome. Fig. 5.6 shows the result of taking three different paths for the vertices not in the faces connected to the cubic corner, the dot marks the cubic corner chosen. Those in the three faces at the cubic corner are obtained directly from the faces. Generally, the 3D objects returned are consistent with a viewer's perception. However, taking average would break the constraint that all the vertices in a face lie in the same plane. But the problem can be rectified by a subsequent correction process, called beautification [69], [70]. The chosen cubic corner strongly influences the result, since all the subsequent computations stem from it. Choosing a grossly inaccurate cubic corner would produce a highly distorted result.

Fig. 5.6. Inaccurate drawings and the recovered 3D objects.

5.4.2 Non- cubic Graphs

We have shown earlier that for objects whose edges and vertices form cubic graphs, it is always possible to visit all the faces. In fact, there are multiple paths to visiting a face. However, for non-cubic graphs, this is not always the case.

The objects in Fig. 5.7a and Fig. 5.7b are not cubic graphs because vertices b and c are of degree four. In Fig. 5.7b, starting from Vertex a as the cubic corner, it is not possible to compute for the coordinates of vertices f and g because it is not possible to obtain the planes of any of the four faces $bcbf$, $defgd$, $cdgc$ and $befb$, as none of them can have more than one known edge prior to their computation. However, in Fig. 5.7a, even though vertices b and c have degree four, vertices f and g can be found, starting from Vertex a as the cubic corner, because the plane of Face $cdgc$, for example, can be found from edges dg and dc . These in turn can be determined from other faces that they belong to, and these faces can be found from the cubic corner.

The distinction between the two cases lies in the fact that Fig. 5.7b contains a non-manifold edge and Fig. 5.7a does not.

As we introduced in Chapter 3, if an object contains non-manifold entities (vertices and edges), it can be decomposed into simpler manifolds along these non-manifold entities [43]. In addition, an object also can be decomposed into simpler manifolds across internal faces [43], [44]. Formally, an object is said to be a simple manifold if it cannot be decomposed into simpler manifolds further. In general, a simple manifold can be recovered while a non-simple manifold cannot, as Fig. 5.7c and Fig. 5.7d further illustrates. The object in Fig. 5.7c, which is a simple manifold, can be recovered. The object in Fig. 5.7d is not a simple manifold as the disconnection of vertices a , b , c and d , which form an internal face, detaches the top pyramid from the base cube, and the apex of the pyramid cannot be determined from a corner at the base. Hence, in general, objects representing simple manifolds, which cover a wider class of objects than cubic graphs, can be recovered fully using our algorithm, but not objects which are not simple

manifolds. Recall that we have introduced an efficient algorithm to decompose an object into simple manifolds in Chapter 3. Therefore, an object which is not a simple manifold can be decomposed into simple manifolds first, and then the cubic corner method applied to recover the individual simple manifolds separately and the results merged to form the original object.

Fig. 5.7. Objects with non-cubic graph.

Whether an object is recoverable can be determined from a face adjacency graph in which the nodes are the faces and an edge exists between two nodes if the two faces are adjacent to each other. This is shown in Fig. 5.8. For ease of illustration, we construct the graph top down, where the top three nodes are the three faces determined by the cubic corner. The nodes of the faces adjacent to these faces are placed below and, similarly, subsequent faces are placed further below. There are two types of edges: directed and not directed. A not-directed edge means neither of the faces it connects to has been determined, except the top three faces at the cubic corner. Initially, all the edges are not

directed. A not-directed edge becomes directed when one face it connects to is determined, pointing from the determined face to the neighbor yet to be determined. Every node whose face can be determined must have at least two edges pointing to it, because every face must be derived from two adjacent faces. The face of a node with less than two edges pointing to it cannot be determined. Using Vertex f as the starting vertex in Fig. 5.7c and Fig. 5.7d gives rise to the graphs in Fig. 5.8. We see that all the faces in Fig. 5.8a can be determined, but not in Fig. 5.8b, as there is only one edge pointing to each of the bottom four faces. A doubly-directed edge exists between the two faces in the middle level because it is possible to compute for either of the faces first and then derive the other face from it – the phenomenon of multiple paths mentioned earlier.

A face that cannot be found from one cubic corner may be found using another cubic corner. But there might not be another cubic corner. Also, even though a majority of mechanical and architectural objects contain cubic corners, there are objects that do not contain any, such as a regular tetrahedron or dodecahedron. Hence it is useful to be able to derive 3D coordinates from non-cubic corners as well.

(a) Graph of object in Fig. 5.7c.

(b) Graph of object in Fig. 5.7d.

Fig. 5.8. Face adjacency graphs for the objects in Fig. 5.7c and Fig. 5.7d; grey boxes for faces that can be found and white boxes for faces that cannot.

Before presenting non-cubic corners, we round off the discussion on non-cubic graphs by looking at the graphs of non-manifold objects. So far, all the objects we have dealt with are manifolds, which are objects with 3D neighborhood homeomorphic to a sphere everywhere. It is possible to derive the planes of all the faces of some non-manifold graphs, such as that shown in Fig. 5.9a. Given the cubic corner a , it is easy to see how to visit every face and obtain their planes. But this is not true of the object in Fig. 5.9b. The plane for Face $bcdeb$ cannot be determined given the cubic corner a .

Hence, depending on its graph structure, it is possible to fully recover a non-manifold object in some cases, and not in others. In other words, some incomplete drawings may be recoverable, some not. Again, in the latter, a solution is to identify more corners, cubic or otherwise.

Fig. 5.9. Non-manifold objects.

5.4.3 Non-cubic Corners

The cubic corner equation gives the coordinates of the end points of the three edges connected to the cubic corner, given the data available in a parallel projection. Here we extend the derivation to non-cubic corners, where the angles between the three edges at

the vertex in 3D can be of any arbitrary values. Suh [63] also provided a derivation for non-cubic corners, but his result is not general, limited to having only one non-right angle.

Let the angles in 3D be A , B and C , as shown in Fig. 5.10a. Let the corresponding angles in the parallel projection be a , b and c respectively, with $a + b + c = 2\pi$ (Fig. 5.10b). To reduce the number of symbols involved, we assume, without loss in generality, that the corner concerned is at the origin, $(0, 0, 0)$. To accommodate a shift in this corner requires a mere translation.

Let $\underline{P}_3$, $\underline{Q}_3$, and $\underline{R}_3$ be the three vectors in 3D incident at the given corner, and $\underline{P}_2$, $\underline{Q}_2$, and $\underline{R}_2$ be the corresponding vectors in the projection plane, which is normal to the direction of projection. Again, without loss in generality, we can let $\underline{P}_3$, $\underline{Q}_3$, and $\underline{R}_3$ be unit vectors and the lengths of $\underline{P}_2$, $\underline{Q}_2$, and $\underline{R}_2$ be α , β and γ respectively, and the projection be in the z direction.

Fig. 5.10. Parallel projection of a non-cubic corner.

Then $\underline{P}_3 = \underline{P}_2 + p\underline{Z}$, (5.9a)

$$\underline{Q}_3 = \underline{Q}_2 + q\underline{Z} \text{ and} \quad (5.9b)$$

$$\underline{R}_3 = \underline{R}_2 + r\underline{Z} \quad (5.9c)$$

where p , q and r are scalars and $\underline{Z}$ is the z direction vector $(0, 0, 1)$. Now, referring to Fig. 5.9a,

$$\underline{P}_3 \cdot \underline{Q}_3 = |\underline{P}_3||\underline{Q}_3| \cos A = \cos A \quad (5.10)$$

since $\underline{P}_3$ and $\underline{Q}_3$ are unit vectors. Also, from Eq. (5.9a) and Eq. (5.9b),

$$\begin{aligned} \underline{P}_3 \cdot \underline{Q}_3 &= (\underline{P}_2 + p\underline{Z}) \cdot (\underline{Q}_2 + q\underline{Z}) \\ &= \underline{P}_2 \cdot \underline{Q}_2 + \underline{P}_2 \cdot q\underline{Z} + p\underline{Z} \cdot \underline{Q}_2 + p\underline{Z} \cdot q\underline{Z} \\ &= |\underline{P}_2||\underline{Q}_2| \cos a + pq \\ &= \alpha\beta \cos a + pq \end{aligned} \quad (5.11)$$

The terms $\underline{P}_2 \cdot q\underline{Z}$ and $p\underline{Z} \cdot \underline{Q}_2$ are both zero because the 2D vectors $\underline{P}_2$ and $\underline{Q}_2$ both lie in the projection plane which is normal to the projection direction $\underline{Z}$. Therefore

$$\alpha\beta \cos a + pq = \cos A \quad (5.12a)$$

Similarly, we can get two more equations involving the other two angles:

$$\beta\gamma \cos b + qr = \cos B \quad (5.12b)$$

$$\gamma\alpha \cos c + rp = \cos C \quad (5.12c)$$

Furthermore, from the projection and by Pythagoras' Theorem, we know that

$$p^2 + \alpha^2 = 1 \quad (5.13a)$$

$$q^2 + \beta^2 = 1 \quad (5.13b)$$

$$r^2 + \gamma^2 = 1 \quad (5.13c)$$

The three relationships in Eq. (5.13) eliminate three of the six unknowns in Eq. (5.12), which is then solvable with three equations and three unknowns. Unlike the case of cubic corners, there are no closed form solutions here, so we have to resort to numerical methods, such as the Newton-Raphson method extended for a system of non-linear equations. From Eq. (5.13), we know that $-1 \leq p, q, r, \alpha, \beta, \gamma \leq 1$. Regardless of which three variables are eliminated in Eq. (5.12), this limited range helps greatly as we can set an initial estimate close to the desired solution.

The required inputs are the six angles. The three in the projection plane, a, b and c , can be obtained automatically from the drawing. The three in 3D space, A, B and C , would need to be specified by the user. From a user interface point of view, this is not always desirable because users do not always know these angles; special efforts are needed to obtain them. But still, these angles are invariant given an object. The alternative is to specify the three end points of the edges at the corner, but this would be very difficult because they vary with the position and orientation of the object.

Ultimately, the solutions of Eq. (5.12) produce the positions of the end points which, in turn, together with the given corner, define the three planes that meet at that corner, which is the desired result.

The derivation for non-cubic corner in perspective and parallel projections are similar, the only one difference is to convert (x, y) coordinates of $\underline{P}_2$, $\underline{Q}_2$, and $\underline{R}_2$ to $(x^*p/f, y^*p/f)$, $(x^*p/f, y^*p/f)$, $(x^*p/f, y^*p/f)$. Therefore, we skip the derivation for non-cubic corner in perspective projection.

5.5 Results

The algorithm has been implemented on a PC in C++. To test the implementation, we created 3D polyhedral objects using a CAD system and display their line drawings using parallel/ perspective projection in an arbitrary view direction. Given an object created in the CAD system, its parallel projection can be obtained easily by dropping the z -coordinates; and its perspective projection is obtained by applying a perspective transformation: $x=X*f/Z$ and $y=Y*f/Z$, where (x, y) is the perspective projection of a vertex of an object, and X, Y and Z are the coordinates of the vertex in 3D read from the CAD system. The CAD data is accurate.

Upon reading in this 2D data, the system invokes the face-finding routine to determine the faces, and then requires the user to indicate a vertex which is a cubic corner. The z -coordinates of all the vertices are then recovered using the algorithm above. These z -coordinates are then compared with the z -coordinates of the corresponding vertices in the original object in the CAD system. They are equal up to ten significant digits. If there are multiple cubic corners in an object, the results are consistent whichever cubic corner is chosen as the starting cubic corner. If there is no cubic corner in an object, an arbitrary non-cubic corner is chosen as the staring corner. Fig. 5.11 shows some test results for parallel projection input drawings; the objects f-n are taken from [71].

(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)

(j)
(k)
(l)
(m)
(n)
(o)
(p)
(q)

Fig. 5.11. Objects recovered from accurate line drawings. In this and subsequent similar pictures of recovered objects, Left: The 2D input, also the recovered object in the input view direction. Middle and right: the recovered object in two different viewing directions.

Fig. 5.12. Objects recovered from (a) two-point and (b) three-point perspective drawings.

Fig. 5.12 shows some results recovered from perspective drawings including two-point and three-point. On the left is the input perspective drawing. The two pictures on the right are recovered 3D object in parallel projection in different views. The dot marks the cubic corner used.

Additionally, our algorithm can be used to recover objects from photographic images.

Lines can be drawn manually on the visible edges of the desired objects, and used as the

input to the algorithm. Fig. 5.13 shows two examples. The picture on the left is the 2D image with lines drawn on the edges; the two pictures on the right are recovered 3D objects in different views.

Fig. 5.13. 3D objects recovered from photographs.

Table 5.1 The length ratio of the recovered object edges compared with the true values.

Length Ratio	True value	Obtained result
Object 1	1.475:1.475:1	1.481:1.469:1
Object 2	1.333:1.093:1	1.324:1.101:1

Table 5.1 compares the length ratios of the key dimensions of the recovered objects at the chosen cubic corner with the values of the objects in Fig. 5.13. The values are very close,

but not quite identical because the input lines, traced by hand, can only be as accurate as the accuracy pixilated graphics offers, which is up to about three significant digits.

5.6 Chapter Summary

This chapter presented a direct method for recovering the 3D objects from parallel/perspective projection. It requires two sets of information from the drawing: the list of faces and a cubic corner. The list of faces can be obtained using an existing algorithm, and the cubic corner has to be identified by the user. The algorithm then computes for all the vertices of the object accurately and directly without iteration. Given the faces, the complexity of the algorithm is linear to the number of faces in the worst case.

The algorithm works for accurate drawings of objects that do not contain non-manifold vertex/edge or internal face. For objects without trihedral vertices, such as the icosahedron, the method is not applicable. Extension to inaccurate drawings was also investigated. A solution for non-cubic corners was also presented.

A number of issues require further work. Firstly, being mathematically exact, the method can be sensitive to inaccuracies in the input: a minor change in the input can result in a large change in the output, which is akin to ill-conditioning in numerical computations. Fig. 5.14 shows the projections of three rectangular blocks with identical front faces of dimensions 1×1 . The three projections look the same to the naked eye, and could indeed be the same in the limited accuracy of pixilated graphics. Actually, they are rectangular blocks of lengths 2, 10 and 20, with their front faces normal to the direction of projection

and then rotated by 2° , 0.4° and 0.2° respectively about the vertical and then the horizontal axis. It is the acute angle of the projection that foreshortens the blocks greatly.

Fig. 5.14. Nearly identical projections of three blocks of different lengths.

Given accurate inputs, our system can recover the correct length of every edge, but the results would be rather counter-intuitive to an uninformed viewer and may not be desirable in the case of inaccurate inputs. Such a phenomenon has been well documented in studies on human vision and 3D shape perception, such as [10]. Edge length preservation is a regularity called “isometry” in Lipson’s optimization-based method for 3D object reconstruction [1].

As Fig. 5.6 shows, we can obtain sensible results for inaccurate inputs, but further investigation is required to ensure the best result, such as in the ill-conditioned situation above and when there is gross inaccuracy in the corner chosen. A hybrid method that is used to obtain better result from inaccurate drawing will be introduced in the next chapter.

Secondly, it is desirable to eliminate the need to identify the cubic corner by the user. Perkins [59] has identified the geometric conditions of the edges at a vertex for a cubic corner to be likely. But ambiguities exist, as the example in Fig. 5.15 shows [10]. Automatic detection of cubic corner remains one of our goals. It is not reasonable,

however, to expect automatic generation of the data required for non-cubic corners; the three 3D angles need to be specified by the user.

Fig. 5.15. Ambiguity: which are the cubic corners, *a & b* or *c & d*?

The method presented in this Chapter is simple and direct for the simple fact that given the geometric and topological constraints that exist in a 2D drawing, the 3D object is completely defined once the coordinates of the first four points at a corner, and hence the three faces that contain them, are known, notwithstanding the cases where some of the faces cannot be reached.

CHAPTER 6

A Hybrid Method for 3D Object Reconstruction from 2D Drawings

6.1 Introduction

This chapter presents a hybrid method to reconstruct 3D polyhedral objects from inaccurate 2D line drawings by combining two known methods, the cubic corner method and the optimization-based method. The cubic corner method can recover accurate 3D objects from accurate drawings and approximate objects from inaccurate drawings. It is deterministic and very efficient, but in the case of inaccurate drawings, the quality of its results varies with the accuracy of the input. Also, there are objects that it cannot recover fully. In general, the optimization-based method produces approximate 3D objects that conform to human perception of the drawing. But it is computationally demanding due to the large search space in the iterative optimization process, and can sometimes converge to incorrect results, partly due to poor initial values for the optimization. The new method starts with the cubic corner method, and then takes its output as the initial guess for the optimization process, which then produces better 3D objects than either method on its own. Our results also show that this new method, though less efficient than the cubic corner method, is more efficient than the optimization-based method operating alone.

6.2 Analysis of the Cubic Corner Method and Optimization-based Method

We have introduced the optimization-based method and the cubic corner method in Chapter 2 and Chapter 5 respectively. In this section, we further analyze the strengths and

weaknesses of the cubic corner method and the optimization-based method, henceforth termed CCM and OBM respectively. The results shown here are from our own implementation of the two methods.

6.2.1 Strengths and Weaknesses of the Cubic Corner Method

CCM works well for accurate line drawings, as Fig. 5.11 shows, returning accurate 3D objects. The vertices of a polyhedron can be derived from the initial four vertices found at the cubic corner, as explained earlier.

CCM may be used for inaccurate drawings also. The quality of the results varies with the quality of the input drawings and the accuracy of the cubic corners used. Fig. 6.1 shows two examples. Fig. 6.1b shows the recovered objects in two different views, based on one specific traversal of the faces given the inputs in Fig. 6.1a. Some of the faces are not planar in the resulting 3D objects, because their vertices are derived from different adjacent faces and hence not guaranteed to lie on the same plane.

Fig. 6.1. Results from inaccurate drawings using CCM.

The coordinates of a vertex are derived from a face to which it belongs. A vertex normally belongs to three or more faces and its coordinates can be derived from any one of them. For inaccurate drawings, the different faces produce different results. Fig. 6.1c shows the objects obtained using a different face traversal order from those of Fig. 6.1b. It can be seen that the resulting 3D objects are different. There is no quantitative measure to say which result is better or if either one is good enough. Different results may also be obtained by choosing a different cubic corner. The effect of an inaccurate cubic corner can be drastic because the derivation of the coordinates of all the vertices stems from it.

In Chapter 5, we suggested that a better result can be obtained by computing for the vertices multiple times from different face traversal orders and return the average of the results. From our experiments, the results indeed look better, as shown in Fig. 6.1d. But they may not be good enough, as can be clearly seen.

Fig. 6.2. CCM with inaccurate inputs (left) and resulting 3D models (middle and right).

Fig. 6.2 shows four further results of CCM with inaccurate inputs and the 3D objects are produced as the average of multiple paths.

6.2.2 Analysis of the Optimization-based Method

Our implementation of the optimization-based algorithm uses six regularities: face planarity, line parallelism, corner orthogonality, skewed face orthogonality, MSDA and isometry. MSDA is the minimization of the standard deviation of angles between lines and isometry is the term used in [1] for the preservation of edge lengths from 2D to 3D. The mathematical details on the regularities can be found in [1]. The weights used in our implementation for these regularities are 1, 20, 20, 80, 100, and 15 respectively. We chose these figures based on our own perception of the relative importance of the regularities in influencing the outcome, tempered by the results of tests we have conducted. Initially, all the unknown z -coordinates are set to zero. Fig. 6.3 shows two test results from our implementation. These results would change if a different set of regularities or weights were used.

Fig. 6.3. Two figures showing results from OBM; the left picture is the input, and the two pictures on the right are the recovered 3D objects in two different views.

The inputs in Fig. 6.3 are two accurate drawings, obtained directly from a CAD system. But the qualities of the results from OBM differ. The result of Fig. 6.3a is good, but approximate, while Fig. 6.3b shows a bad result. Here, “good” means that the result is visually close to what a human would expect, and “bad” means that the result is obviously not so. These are qualitative terms which can be subjective. In Section 6.3, we will compare the results using a quantitative and objective measure based on objective function values.

Computationally, this iterative optimization-based method suffers from slow convergence and is susceptible to arriving at a local minimum, which is why it can deliver poor results.

The conclusion is that the optimization-based method produces good results at times and poor results at times. The user has little control in influencing the outcome. Even if a system is programmed such that users are allowed to choose the regularities and the weights, it is difficult to see how they can know what regularities or what weights to choose for a specific input to produce the best results. It would be a matter of trial and error which is not desirable. It is also possible to give the user control over the initial values of the z -coordinates to help the optimization process, but this is again difficult as there can be many vertices and therefore many z values to define. Requiring more user inputs defeats the purpose of automatic 3D reconstruction.

6.2.3 Summary

The conclusion of the analysis is that neither CCM nor OBM can guarantee desirable results consistently in general, though CCM can for accurate inputs. The idea of combining the two methods to harness the strengths of both seems a good proposition.

6.3 A Hybrid Method for Inaccurate Drawing

We now present a hybrid method for inaccurate drawings, combining CCM and OBM. It overcomes the deficiencies of both CCM and OBM, at a higher computational cost than that achievable by CCM but well below that of OBM. This method can be simply stated in the flow chart in Fig. 6.4.

Fig. 6.4. Flow chart of the hybrid method.

We have shown that, using CCM, we may obtain results which are not necessarily acceptable when the input 2D line drawings are not accurate. But they are good approximations to the target outcome, and can serve very well as the initial input to OBM.

This idea of giving OBM a better initial input than simply setting the unknown z -coordinates to zero has been reported before. Without any mechanism to generate a good initial set of values, setting the z -coordinates to zero is an easy way out, as is done in [19]. Given this uninspired guess, the approach is susceptible to converging to a local minimum in the optimization process, which means that it produces an implausible result. Lipson and Shpitalni [1] and Company et. al. [57] provided mechanisms which will be discussed in Section 6.4.2, to produce a better initial guess.

6.3.1 The Importance of the Initial Guess for the Optimization

Setting an initial guess is the first step of any iterative procedure. Iterative algorithms are sensitive to the initial guess because the distance of this guess from the target influences greatly the convergence (or not) to the target and the computation time.

Take a simple one dimensional root seeking problem as example, the number of steps to reach the target is increased when we enlarge the difference between the initial guess and the target. For a high dimensional optimization problem, this difference can amount to a huge increase in computation time and affect the final outcome. For our problem of finding the z -coordinates of the vertices, the dimension is n , the number of vertices in the drawing. The complexity of the optimization process is $O(kn^3)$, where k is the number of regularities used [1]. When n is large, a small change in the initial guess can result in a large increase in the computation time.

It is difficult for the user to specify an appropriate initial guess for the optimization because, even when we have some physical insight to guide us, the large number of variables in the objective function is a deterrent, as is the case when there are many vertices, and hence many z -coordinates, in a drawing. Also, the search range of $-\infty < z < \infty$ is large for every z . Furthermore, as stated earlier, this would defeat the purpose of an automatic 3D shape reconstruction system.

6.3.2 Testing Procedure

The hybrid method has been implemented in C++ and runs on a Pentium dual core PC, with two gigabytes of memory at 2.4 gigahertz clock speed. It is a mere combination of the CCM and OBM procedures that have been implemented separately, according to the flow given in Fig. 6.4. Minor modifications are required to the CCM procedure to cater to objects that CCM cannot recover completely, as discussed later in Section 6.3.4. Our implementation has been tested extensively. The tests are conducted for drawings of different objects. For each object, we use drawings with varying degrees of inaccuracy. As mentioned earlier, we first used a CAD system to create a 3D model, which is then

projected on to a plane. The input drawing is this projection with the z -coordinates of the projected vertices removed. The remaining x and y coordinates are exact up to the accuracy of double precision computation. We then create inaccurate versions of this drawing by perturbing the x and y coordinates, by adding to them small randomly generated values. The degree of inaccuracy is controlled by setting the maximum value of this perturbation. We define this inaccuracy value as δ_{max}/d , where δ_{max} is the maximum displacement of a point from its accurate position and d is the length of the diagonal of the tight bounding box of the accurate drawing aligned to the x and y axes.

6.3.3 Results

This section shows the results of reconstruction for seven different objects; those in Fig. 6.5-6.7 are our own creations and those in Fig. 6.8-6.11 are modeled on drawings of other authors. For each object, we show results from four drawings at different inaccuracy values, the first of which is always 0.0, meaning that the inputs are exactly as generated from the CAD system. For each drawing, we show the recoveries from CCM, OBM and the hybrid method.

Provided with the displays of each object are quantitative data associated with the reconstruction. For objects with known 3D geometry, the quality of a recovered object can be measured against the known data. When no known geometry is available, there is no good quantitative measure for the quality. There is our own visual perception and judgment, but that is only qualitative. One possible quantitative measure is the final objective function value (OFV in short) produced by OBM. This value is a direct outcome of OBM and the hybrid method. For CCM, the value is obtained by simply

applying the objective function to the object it produces, using the same set of regularities and weights, without performing any optimization.

The actual value produced by the objective function is not important, as it varies with several things, including the number and size of the input coordinate values. The more or bigger these values, the bigger would be the OFV. Hence an object centered around the origin would have a smaller value compared to the same object centered at, say, (1000, 1000, 1000). This value also depends on the regularities and the associated weights applied. Therefore we cannot compare the OFVs of different objects to determine if one object is better than another. The OFVs are only meaningful for comparing the quality of the same object produced by the three methods; they are a quantitative means of comparing the outcomes. The OFVs need to be taken with caution even for the same object but at different accuracies because it is not guaranteed that the contributions to each regularity are from the same set of geometric entities when the accuracy changes. For example, two parallel lines contributing to the parallelism regularity may fall outside the parallelism threshold when the inaccuracy increases, and thus are not included in the parallelism computation any more. This means that, if we compare the OFVs produced by the optimization process using two drawings of different accuracies, we may not be comparing the same thing. When the differences are sufficiently large, the drawings could be as good as those of different objects. Hence, comparison of quality using OFVs should be made only for the three different methods using the same input, though in general, it can be said that the lower the OFV, the better is the recovered object. That is because, in the way it has been implemented, the OFV is the weighted sum of the

contributions from regularities which return a lower value when there is a better compliance between 2D and 3D.

In our tests, the accuracies are chosen such that the drawings are not so far distorted that they bear little resemblance to the original.

The ultimate target is to attain an OFV of zero, but this is unlikely because it is not usual that all the regularities can be satisfied simultaneously, even for perfectly accurate input drawings and perfectly reconstructed 3D objects. For example, two lines that are seen to be parallel in 2D may not actually be parallel in 3D, and hence the function accounting for these two lines being parallel would not return a zero value.

In Fig. 6.5, it can be seen that CCM produces good results when the accuracy is good. But when the accuracy is poor, the result becomes unacceptable; the 3D object is too distorted and the OFV is high. In the fourth case, the object produced by CCM bears little resemblance to the drawing, because some coordinate values that it returns are far from the desired values, as this mathematically precise method does not attempt to preserve edge length, as explained in Section 6.2.1. The results of CCM are produced as the average of three results obtained by three different traversal orders of the faces.

The first three OBM results are good, both visually and in terms of the relatively low OFV. The fourth is relatively poor, but that is not a surprise given the distortion in the input.

2D Input	CCM	OBM	Hybrid
0.0	734, 0.099 ms	5113-582, 2500 ms	734 -670, 1344 ms
0.009	680, 0.099 ms	5029-712, 2437 ms	680-663, 1219 ms
0.054	2304, 0.100 ms	5162-848, 2359 ms	2304-701, 1688 ms
0.089	1696, 0.123 ms	5963-1912, 2109 ms	1696-1174, 1468 ms

Fig. 6.5. Four line drawings at four different accuracies and the recovered objects using CCM, OBM and the hybrid method. On the left is the input drawing. The number below the drawing is the accuracy level. The dot in the drawing marks the cubic corner used.

Under each method are two pictures of the recovered 3D object from two different viewpoints. Under CCM, the first number is the OFV of the recovered object, and the second number is the time taken to recover the object, in milliseconds. Under OBM and Hybrid, the first two numbers are the initial and final OFVs, and the third number is the reconstruction time, in milliseconds.

For the accurate drawing, the hybrid method is able to produce a “better” result than CCM in the sense that the OFV is lower, which seems surprising since the CCM result is accurate. That is because the hybrid method takes the CCM output and puts it through OBM, which then attempts to reduce the value further. OBM only accepts results with lower OFV. When the result from CCM is accurate, i.e. it matches the original 3D model, a lower OFV from OBM does not imply a more accurate result, as that is not possible. An

exact 3D model does not necessarily give rise to a low OFV, since that depends on the regularities and the geometry of the object. It is quite possible that some shifts in the vertices can produce a lower OFV, and that is what OBM has done, producing an object that deviates from the accurate model.

2D Input	CCM	OBM	Hybrid
0.0	291, 0.070 ms	2976-2763, 1094 ms	291-271, 375 ms
0.010	332, 0.082 ms	3072-2771, 1344 ms	332-280, 391 ms
0.057	404, 0.068 ms	2563-2051, 1234 ms	404-322, 594 ms
0.095	1423, 0.082 ms	2721-524, 603 ms	1423-995, 879 ms

Fig. 6.6. The recovery of another object. Please see Fig. 6.5 for the explanation of the numbers.

The time for CCM to recover an object is consistently very low, around 0.1 ms, whereas OBM typically takes well over 10,000 times longer. The hybrid method takes more time than CCM but less than OBM in general, because it starts with an initial model that is closer to the final solution than what we can set for OBM.

It should be noted that all three methods attempt to recover an object that best fits the actual input drawing, not the accurate version, as there is no notion of an “accurate version” as far as the computational processes are concerned.

In Fig. 6.6, the results of CCM are good for the first three drawings but poor for the last, as is visually obvious. OBM produces three poor results but a better one for the least accurate drawing. This highlights one characteristic of OBM, that its results are unpredictable and inconsistent. Again, the hybrid method produces consistently better results, except for the last one, where OBM is better and faster. This may point to the fact that, when a drawing is highly inaccurate, there is no true 3D object that fits the projection closely, and the 3D approximation produced by CCM is no better an initial input to OBM than simply setting all the unknown z -coordinates to zero. Alternatively, it could simply be that in this particular situation, OBM simply requires more searches before convergence for the input offered by CCM.

In Fig. 6.7, all three methods consistently deliver good results, though the OFV for CCM is higher for the last two drawings. Again, this is not surprising since CCM does not work to reduce the OFV. What may be surprising is that, despite having a worse initial input, OBM delivers a better OFV than the hybrid method for the last drawing. This can be again attributed to the unpredictable nature of OBM in its ability to converge to a local minimum. Another point of note is the existence of two k-shape vertices (where there are two collinear edges at a vertex of degree four) in the drawing, demonstrating the ability of the three methods to handle objects beyond simply trihedral objects.

2D Input	CCM	OBM	Hybrid
0.0	445, 0.115 ms	3481-429, 2281 ms	445-399, 1047 ms
0.010	401, 0.114 ms	3458-414, 2375 ms	401-396, 1219 ms
0.060	898, 0.134 ms	3816-734, 2625 ms	898 -645, 1515 ms
0.099	1239, 0.116 ms	4320-869, 3090 ms	1239-906, 1455 ms

Fig. 6.7. Another example. Please see Fig. 6.5 for the explanation of the numbers.

Fig.6.8 is an example from Lipson and Shpitalni [1]. Here CCM exhibits poorer results when inaccuracy increases, so much so that the resulting object looks unrecognizable in the last two cases. The hybrid method consistently improves them. But when the inaccuracy is too high, as in the fourth case, even the hybrid method cannot produce a good enough result. The hybrid method takes more time than OBM in the last two cases, signifying again that, in case of high inaccuracy, the initial input to OBM offered by CCM may not be better than assigning all the unknown z -coordinates to zero.

2D Input	CCM	OBM	Hybrid
0.0	780, 0.09 ms	4202-979, 1312 ms	780-615, 473 ms
0.01	947, 0.09 ms	4207-773, 1219 ms	947-612, 576 ms
0.07	2887, 0.09 ms	4667-1032, 1063 ms	2887-1652, 1024 ms
0.11	3869, 0.08 ms	4856-1583, 1165ms	3869-1524, 1298ms

Fig. 6.8. An example from Lipson and Shpitalni [1]. Please see Fig. 6.5 for the explanation of the numbers.

A line drawing of a cube showing all the edges has two interpretations, commonly known as Necker's Cubes [59]. In fact, a line drawing of any object showing all the edges has two interpretations. For symmetric objects, the two resulting objects are the same. For non-symmetric objects, they are different; they are laterally transposed, that is, mirror image of each other.

The cubic corner formula provides for the two interpretations directly [59, 72-75]. So CCM is able to recover the two objects separately, as Fig. 6.8 and 6.9 illustrate. This means that the hybrid method can too, given the initial model from CCM. But this cannot be done using OBM on its own, as there is no known way to direct the optimization process to produce one or the other result; it can only produce one of them, and we

cannot predict which. That is why the results of OBM in both Fig. 6.8 and 6.9 are the same.

2D Input	CCM	OBM	Hybrid
0.0	780, 0.09 ms	4202-979, 1312 ms	780-615, 473 ms
0.01	946, 0.09 ms	4207-774, 1219 ms	946-612, 576 ms
0.07	2887, 0.09 ms	4667-1033, 1063 ms	2887-1652, 1024 ms
0.11	3870, 0.08 ms	4856-1583, 1165 ms	3870-1524, 1298 ms

Fig. 6.9. The same inputs as in Fig. 6.8. The results for CCM and hybrid are laterally transposed from that of Fig. 10, but the result for OBM is unchanged. Please see Fig. 6.5 for the explanation of the numbers.

Fig. 6.10 is an example due to Varley and Company [55]. The results here are consistent with those of Fig. 10. A point of note is the two vertices with six incident edges, demonstrating that the methods work for objects with vertices of higher degree.

2D Input	CCM	OBM	Hybrid
0.0	460, 0.084 ms	6401-2060, 797 ms	460-329, 640 ms
0.009	502, 0.083 ms	6377-1729, 719 ms	502-329, 625 ms
0.056	2704, 0.082 ms	6516-1539, 688 ms	2704-641, 906 ms
0.093	2896, 0.084 ms	6875-981, 813 ms	2896-986, 732 ms

Fig. 6.10. An example due to Varley and Company [55]. Please see Fig. 6.5 for the explanation of the numbers.

Fig. 6.11 is derived from an object found in [55], where the shape is a hexagonal cylinder of uniform cross sections. We modify it by tapering the cylinder and the two slots at the top and the bottom. This results in an object with no cubic corner. This demonstrates CCM's ability in dealing with non-cubic corners [72]. Again, CCM produces good results when the accuracy of the input is good, but rather poor results otherwise. OBM, on the other hand, produces erratic results with one good one amongst three poor ones. The hybrid method improves on both results in general.

2D Input	CCM	OBM	Hybrid
0.0	568, 0.116 ms	3238-1275, 1516 ms	568-564, 593 ms
0.008	608, 0.106 ms	3226-1265, 2125 ms	608-574, 625 ms
0.050	1046, 0.104 ms	3494-757, 1328 ms	1046-994, 1375 ms
0.081	1338, 0.104 ms	4112-2076, 1391 ms	1338-1127, 1281 ms

Fig. 6.11. Another example due to Varley and Company [55]. Please see Fig. 6.5 for the explanation of the numbers.

6.3.4 Objects Which CCM Cannot Fully Recover

As pointed out in Section 5.4.2 of Chapter 5, there are some drawings that cannot be fully recovered using CCM. This is dealt with easily using the hybrid method. The drawing is first put through CCM to recover the vertices that it can recover. A vertex that cannot be recovered is then given some initial value; in our implementation, this value is the average of the vertices the unknown vertex is directly connected to. This new set of vertices is then put through OBM, which then produces an object which conforms to human expectation. Fig. 6.12 shows an example.

2D Input	CCM	OBM	Hybrid
0.0	3172, 0.128 ms	3728-1652, 1531 ms	3172-899, 640 ms
0.009	1505, 0.115 ms	3730-1580, 1219 ms	1505-918, 797 ms
0.053	2441, 0.115 ms	3823-2174, 937 ms	2441 -1062, 656 ms
0.089	2896, 0.116 ms	4052-1880, 813 ms	2896-1212, 828 ms

Fig. 6.12. A non-simple manifold which CCM cannot fully recover. Please see Fig. 6.5 for the explanation of the numbers.

The object in Fig. 6.12, again modelled on an object found in [55], is a non-simple manifold, which means that the four vertices at the top cannot be recovered by CCM if we start from a vertex in the lower body. Our system proceeds to recover all the vertices it can and then estimates those that it cannot. As can be seen in the column under CCM, the resulting object and the OFV are both poor. OBM can obtain better results in this case. But when the CCM output is passed to OBM as the initial value for the optimization, the outcome is much better, as shown in the Hybrid column.

Fig. 6.13 shows two more results of the hybrid method. Note that the second object is non-manifold, where the geometry of the dangling faces is not recovered by CCM, but by OBM.

Fig. 6.13. Objects with inaccessible vertices in CCM. These vertices are marked by dots and the cubic corner is labeled *a* in the input drawing on the left. The two pictures on the right show two different views of the 3D object recovered by the hybrid method.

6.4 Discussion and Summary

In these discussions, we focus on two issues: the operational domain of the method and its performance compared to other existing methods, followed by the conclusion.

6.4.1 Operational Domain

As noted earlier, CCM works well for parallel projections, and the parallel projection must be orthographic. Oblique parallel projections, including cavalier projections, require further inputs to specify the orientation of the projection plane and modification to the cubic corner formula. Our current implementation does not cater to that and hence cannot deal with oblique parallel projections.

The cubic corner formulation can be extended to perspective projection, and this has been introduced in Chapter 5. Li et. al. separately reported on a different closed form method to recover objects in perspective drawings [26]. An initial attempt has been reported by Fong and Cham [23] to extend OBM to perspective projection, but the work is demonstrated only in the limited domain of simple collections of rectangular blocks. As yet, the hybrid method cannot work for perspective projection in general. This is because OBM cannot deal with perspective projection.

As we mentioned in Chapter 5, CCM works fully for simple manifold objects. CCM also works fully for non-manifold objects where there is a path for the traversal of the faces such that the plane of every face can be established. For objects CCM cannot deal with fully, the hybrid method comes into its own, where OBM is invoked to recover the vertices CCM cannot reach. This includes dangling edges and dangling faces, as demonstrated in Section 6.3.4.

6.4.2 Comparison with Reported Works

Numerous pioneers have produced results in the reconstruction of 3D objects from 2D drawings. Some employ purely one technique. For example, Lamb and Bandopadhyay's method [75] is akin to CCM, where it systematically works through the edges of the object to obtain the coordinates for the vertices, but it operates in a very limited domain of largely orthogonal objects. There is no result in [75] that can be used for comparison. Another example is that of Leclerc and Fischler [19], who employs purely the optimization-based method using only two regularities, MSDA and planarity, and setting all the unknown z -coordinates to zero initially. This work is encompassed within our implementation of OBM, which uses six regularities, including MSDA and planarity. The

types of result obtainable from the method described in [19] and similar works can therefore be glimpsed from the results of our OBM, and explicit comparison is unnecessary.

There are two works that uses the pipeline that we employ in our hybrid method: that of establishing an initial model and then feeding it to OBM. The first is reported by Lipson and Shpitalni [1], where they use an angular distribution graph to help produce a tentative model. Impressive results are given in [1], but there is no indication of the improvement due to the tentative model. A visual comparison can be made by referring to our Fig. 6.8 and 6.9, which use an example in [1]. Both CCM and the hybrid method produce results that are visually as good as that shown in [1]. No quantitative comparison is possible.

The second is by Company et. al. [57], who require two methods for establishing the tentative model, one for the restricted domain of normalons² and quasi-normalons, and the other, called level inflation, applies to general polyhedra. The former can produce exact solutions on its own, but only within the said restricted domain. Level inflation can only produce approximations not suitable as final results. Our hybrid method is different in that CCM is a standalone method on its own, capable of handling polyhedra in general, including those without cubic corner, as the example in Fig. 6.11 testifies. Company et. al. stated that “even the simplest optimization algorithm with a poorly defined objective function should find the global optimum provided it starts with a good tentative model”, which we find to be true for accurate inputs (except that we do not consider our objective function to be poorly defined), but the result is inconsistent for inaccurate inputs.

² A normalon is a polygon having the property that the angle between any two of its adjacent sides is 90° .

One potential weakness of our hybrid method is its dependence on finding the correct faces from the drawing. We have introduced a new face finding method in Chapter 4, and the method always returns correct faces, except in the very unusual situation of connected internal faces adjoining holes. Hence the hybrid method has a low possibility to fail. Company et. al. [57] does not rely on faces to obtain their initial model, so face finding is not an issue here. But it would be if they require face planarity as a regularity in the optimization process.

Most 3D reconstruction publications do not deal with the phenomenon of Necker's Cubes, other than a mention of their existence. Optimization-based methods, because of their inability to discriminate between the two solutions, can only offer one as the result, which might not be the desired one. However, this is but a minor flaw since the alternate result can be obtained by simply negating the z -coordinates obtained [3].

6.4.3 Chapter Summary

The hybrid method presented in this paper for inaccurate drawings is the combination of the cubic corner method and optimization-based method. It overcomes the deficiencies of both CCM and OBM to produce a much better result than that attainable by either method alone. It takes advantage of the efficiency of CCM to obtain an initial 3D shape from the input line drawings. This serves as the initial guess for OBM, which then returns a better 3D object. Because the initial guess from CCM is normally fairly close to the final outcome, the search range of the iterative optimization process is much reduced, thus reducing the amount of iterations in OBM and the computation time. Because the time spent in CCM is almost negligible, the total time taken for the two processes is

usually a lot smaller than OBM alone. Furthermore, the probability of convergence to the expected object in OBM is much improved also.

The significance of this hybrid method lies in the consistently high quality results that it delivers, at a much reduced time compared to OBM. The quality of the results delivered by CCM and the hybrid method varies with the accuracy of the input drawing; the more distorted the drawing, the worse the result. As can be seen in the results in Fig. 6.5-6.11, the consistency of the results from OBM is poor; it returns good results at times and poor results at times. It can produce unacceptable results even when the drawing is accurate. Company et. al.[57] observed that “Optimization is a good approach, but still too academic to be used as an engine for a sketch-based modelling system, since it is too sensitive to the imperfections inherent in the sketches.” We agree in regards to its role as an engine for a sketch-based modelling system, but the erratic results in our tests with both accurate and inaccurate inputs lead us to conclude that it is not only imperfections in the sketches that cause it to fail, but the propensity of the optimization process to settle at a local optimum.

The question remains as to whether OBM alone can do a good job if a more comprehensive set of regularities is used instead of the set of six used in our implementation. Furthermore, there is also the question of whether using different weights in association with the different regularities would make any difference. Our contention is that, whatever optimization algorithm one uses, because of its very nature of seeking improvement to a previous solution by making incremental changes to the unknown variables, the solution arrived at will always remain the “best guess”, whatever the regularities or weights. A more comprehensive set of regularities will help but,

inevitably, there will still be cases where it converges to a poor result. A helping hand from CCM with a better initial value can only improve matters.

To keep the focus, this chapter has concentrated mainly on objects with cubic corners. But as demonstrated in Fig. 6.11, CCM can also be applied to recover objects that have no cubic corners. This means that the same hybrid method can be applied to objects with no cubic corners also.

The hybrid method also overcomes the problem of CCM where it is not able to recover some objects fully because of the inability to access some vertices. But one type of object remains unrecoverable: those that contain no trihedral vertices, such as the icosahedron, as CCM requires at least one trihedral vertex. Fortunately, such objects are rare.

CHAPTER 7

3D Reconstruction by Reverse Engineering of Drawings in 2-Point Perspective View

7.1 Introduction

This chapter presents a novel algorithm to reconstruct the 3D object depicted in an accurate 2D line drawing in a 2-point perspective view. It reverses the technique used to draw perspective views of 3D objects taught in drawing classes, which creates a perspective view from two orthogonal planar projections. Our method first recreates the plan view of the object, which gives the x and y coordinates of the vertices, and then obtains the height of each vertex separately by recreating the side projection. The plan view and the height together give the x , y , z coordinates of the vertices, which define the object in 3D. The method reconstructs the object directly from the line drawing, does not normally require face information, and is highly efficient.

The reader may well ask: “Given the cubic corner and the hybrid method reported earlier, why present another method?” The reason is simple. This method was one of the very first the author developed in her PhD research. Though it is more restrictive, applying only to two-point perspectives, the technique, exploiting the interesting notion of reverse engineering, is very different and very efficient and is thus worth reporting and archiving.

7.2 Outline of the Algorithm

Given a 2D perspective line drawing, our method has the following four stages for 3D reconstruction: framework construction, plan view construction, height determination and 3D reconstruction.

In the first stage, some properties of the perspective world in the drawing are defined. The second stage gives us the x and y coordinates of the vertices and the third stage gives us the z coordinates. The last stage defines the object in 3D.

7.3 Framework Construction

The key in our method to recover the 3D object in a perspective drawing is to obtain the plan view and height of each vertex of an object. To do so, we need to determine some properties of the perspective world in the drawing: the horizon, ground line, picture plane and station point (see Fig. 7.1). Here, we use the definitions of these four items in [76], as follows:

- **Picture plane:** imaginary transparent plane onto which the object is being projected.
- **Station point:** center of projection.
- **Ground line:** intersection of the picture plane and the ground plane which is at 90 degrees to the picture plane and is used to place the object.
- **Horizon:** intersection of the picture plane and the horizon plane parallel to the ground plane.

Given a two-point perspective drawing, we need to find its vanishing points first. Parallel lines in 3D converge at a point, the vanishing point [77]. Our method requires two principal vanishing points from lines perpendicular to each other in 3D. The left vanishing point (LVP) and the right vanishing point (RVP) are marked when a designer starts to draw a picture. Therefore, they are fixed parameters in our algorithm. The horizon in a two-point perspective drawing is a horizontal line on which both vanishing points lie. Therefore, it can simply be determined by connecting the two known principal

vanishing points. Without losing generality, we make the assumption that the object in the picture touches the picture plane in 3D space. This assumption ensures that the plan view of the object will touch the picture plane. The ground line is defined to be a line parallel to the horizon and goes through the lowest point touching the picture plane. We term the lowest point touching the picture plane the ground point (GP).

The locations of the station point and the picture plane are related. We must first fix one of them, the other can then be determined. Here, we locate the picture plane first. This location is arbitrary; we only need it to be parallel to the horizon in a two-point perspective drawing [78]. Its location depends on where we want to see the plan view of the object depicted in a perspective drawing.

It is important to determine the position of the station point because it plays a key role in determining how the viewer sees an image. For a two-point perspective drawing, the vanishing points for any 90° angle in space form a 90° angle at the viewer's eye. Therefore, one constraint on the location of the station point is that it must lie on the semicircle with the line joining the two vanishing points as diagonal (see Figure 1b), as this ensures the 90° angle. The reconstructed 3D object from any point on the semicircle could be accepted as correct [79]. Even though the viewer's eye position (i.e. the station point or the center of projection) is not unique, its projection on the image plane is more likely to be positioned at the center of the image [65] because, in the real world, the camera is considered to be positioned pointing toward the center of the object. Therefore in this chapter, we assume that the projection of the station point on the image plane coincides with the center of the image.

Fig. 7.1a shows a two-point perspective view in 3D and Fig. 7.1b shows its plan view, where LVP and RVP stand for left vanishing point and right vanishing point respectively. Now, we can fix the position of the station point based on the constraint and our assumptions discussed previously. From Fig. 7.1b,

where x_{MAX} and x_{MIN} are the maximum and minimum x coordinates of the image respectively; l and r are the respective horizontal distances of the left vanishing point and the right vanishing point to the station point; x_{SP} and y_{SP} are the x and y coordinates of SP and y_{PP} is the y coordinate of the picture plane.

(b) Two-point perspective in spread out plan view

Fig. 7.1. Two-point perspective view in 3D and in spread out plan view.

7.4 Obtaining the Plan View

We have created the framework of the system consisting of the horizon, ground line, picture plane and station point. To do the next step, seeking the plan view of the object, it is necessary to project all the vertices and hence the line segments of the object onto a plane which is parallel to the ground plane, we call it the bottom plane. All vertices or line segments of the object are projected on to the bottom plane and the projection is in the vertical direction which is perpendicular to the bottom plane. We select one vertex touching the ground line as the ground point, or GP. The bottom plane contains GP and two lines, LVP-GP and RVP-GP. Before we perform the projection, we need to define several terms:

L_L : a line segment passing through the left vanishing point on the horizon.

R_L : a line segment passing through the right vanishing point on the horizon.

V_L : a line segment in the z direction, perpendicular to the ground line.

G_L : a general line segment which is none of the above three.

V_V : the set of all the vertices in the 2D line drawing.

7.4.1 Vertex Projection onto the Bottom Plane

The coordinates of the vertices in the bottom plane are their projected coordinates. Hence the first act of the algorithm, upon establishing the bottom plane, is to find these vertices. Their projected coordinates are the initial cues in our algorithm. Lines and vertices not in the bottom plane need to be projected onto the bottom plane, in order to determine their x , y coordinates. To do the projection, we utilize four rules:

Rule 7.1. The projection of a line parallel to the bottom plane must be parallel to the line itself.

According to the Rule 7.1, if a line is a L_L or R_L , its projection on the bottom plane must also be a L_L or R_L .

Rule 7.2. The projections of the two endpoints of a V_L are the same point on the bottom plane.

Rule 7.3. The line joining a vertex and its projection on the bottom plane is a vertical line.

Here, we use a simple example to illustrate using these three rules.

A vertex is the end point of two or more edges. By traversing the edges systematically, the position of the vertices in the bottom plane can be determined one by one if the object does not contain any G_L . In reality, most objects contain G_L , which may result in undeterminable vertices; a vertex is undeterminable if its projection on the bottom plane cannot be determined by only using the above three rules. A G_L cannot be projected onto the bottom plane directly because it has no specific relationship with its projection on the bottom plane. Therefore, we must resort to other information, such as information from an adjacent edge or face, to project two end points of a G_L onto the bottom plane.

The object in Fig. 7.2 contains four G_L s; every one of these four edges shares common vertices with other non- G_L s. So, the two end points of any G_L can be projected by using the above three rules, because they also belong to non- G_L s. In turn, these G_L s can be projected onto the bottom plane. But, not all G_L can be projected by using this method. Take the objects in Fig. 7.3a and b as examples, the line segment l_0 is a G_L , and both its end points are connected to non- G_L . But, a non- G_L on the top face cannot be determined because none of its end points is projected, so l_0 also cannot be projected on to the bottom plane, because one of its endpoints is on the non- G_L . A line that cannot be projected onto the bottom plane using the above three rules is said to be isolated. We need to use another cue, face information, to project isolated lines.

This means that there is a need to identify the faces of an object from the drawing to handle isolated lines. Face identification is a difficult but well studied problem, and a number of solutions have been developed [45], [54].

Once we have the face information, the vertical faces containing vertical lines can be found. Since all the vertices are projected onto the bottom plane in the vertical direction, all the lines of a vertical face will be projected as a line onto the bottom face. An isolated line can be projected onto the bottom plane if it is on a vertical face and at least one non- V_L of the face has been projected on the bottom plane. Because a non- V_L 's projection on the bottom is a line, all vertices on the vertical face containing the non- V_L can be projected onto the same line. A V_L 's projection on the bottom is a point; other lines of the vertical face containing the V_L still cannot be projected on the bottom plane if only the V_L has been projected. In Fig. 7.3a, the vertices i_0 and i_1 belong to a vertical face, and there exists a non- V_L projection segment, marked in bold, in this vertical face. Combining this condition with Rule 7.3, i_0 and i_1 can be uniquely projected onto the bottom plane. The other vertices on the top most face can all be projected by using the above three rules. The top most vertex i in Fig. 7.3c also can be projected with the help of face information, and its projection on the bottom plane lies on the bold line, since the vertex and this line belong to the same vertical face.

Fig. 7.3. Projection onto the bottom plane by using the three rules and face information.

For the object in Fig.7.3b and d, all the vertices on the top most face cannot be projected because none of the side faces that contain them are vertical, meaning that the vertices on

these faces and their projection on the bottom plane have no specific geometric relationship. Therefore, the limitation of our method: the inability to handle objects which contain G_L with undeterminable vertices because of insufficient information. An extra piece of information could help. For example, if the vertex i of the object in Fig. 7.3d is symmetrically positioned, then, its projection can be located at the center of the bottom rectangle.

7.4.2 Drawing the Plan View

Our method recovers the 3D object by reversing the process which produces the perspective drawing based on two orthogonal views of the object. After projecting all the lines onto the bottom plane, to get the plan view of the object, the next thing is to draw all the projected lines on the picture plane. According to the principle of perspective drawing [77], “sight lines” pointing parallel to the object lines are drawn to locate the vanishing point on picture plane. This perspective drawing principle is demonstrated using the example in Fig. 7.4.

Fig. 7.4. An illustration of the process of drawing the plan view.

Given a line, say L , of a perspective drawing, its sight line can be defined as the line joining SP to the point which is the projection of the vanishing point of L on the picture plane. Our method gets the plan view of the projected bottom plane by reversing the perspective drawing process. Fig. 7.4 shows, for a given two-point perspective drawing, with the vanishing points given by the designer, how the sight line of each line segment in the object is determined. For example, the sight line of L_0 is $SP-P_0$, and the sight line of L_1 is $SP-P_1$. According to the perspective drawing principle [80], the sight line of a line in a perspective drawing must be parallel to the line in the 3D object. With the sight line of a line in perspective drawing known, the direction of the line in 3D, and therefore in the plan view can be determined, since the line in 3D must be parallel to its sight line. Hence we know that the lines corresponding to L_0 and L_1 in the plan view are parallel to $SP-P_0$ and $SP-P_1$ respectively. Similarly, the direction of each line in the plan view can be determined. To get the accurate location of each line in the plan view, we utilize the following two important properties of perspective drawing:

Property 7.1: A line located on the bottom plane and its corresponding line in the plan view have the same projection on the picture plane.

Proof: A line located on the bottom plane and its corresponding line in the plan view are the same line in 3D. The only difference between them is the view chosen; the picture plane remains unchanged regardless of the view point one looks at the object from, so their projections on the picture plane must be the same. ■

Property 7.2: The intersection between the ground line and a line L located on the bottom plane and the intersection between the picture plane and the line L in the plan view have the same projection on the picture plane.

Proof: Take a line L located on the bottom plane and its corresponding line in the plan view L' , we know that L and L' are the same line in 3D. We also know that the ground line is the intersection of the ground plane and the picture plane, and the ground line is the projection of the picture plane on the ground plane in the vertical direction. So, the ground line and the picture plane actually are the same line in the plan view. Hence, the intersection between the ground line and L equals to the intersection between the picture plane and L . Since L and L' are the same line in 3D, therefore, intersections between them and the picture plane coincide in the plan view (see Fig. 7.1a). ■

This property is used to fix the location of L' . Take the line L_1 in Fig. 6.4 as an example, its intersection with the ground line is T_1 . The projection of T_1 on the picture plane is TP_1 . According to Property 7.2, the corresponding line of L_1 in the plan view must go through TP_1 .

To summarize, given a line, L , of a perspective drawing, the perspective drawing principle provides us with the direction of its corresponding line L' in the plan view. Property 7.2 gives the location of L' and Property 7.1 gives us the exact coordinate of the two endpoints of L' .

Now, we can use the perspective drawing principle and the above two properties to get the plan view of each line on the bottom plane. Assuming a line $L_1: y=kx+b$ on the bottom plane and its corresponding line in the plan view is $L_1': y=k'x+b'$. The vanishing

point of L_1 on the picture plane is $P_L (x_{PL}, y_{PL})$ and station point is SP (x_{SP}, y_{SP}) .

According to the perspective drawing principle, we have,

$$k' = \frac{y_{PL} - y_{SP}}{x_{PL} - x_{SP}} \quad (7.4)$$

Based on Property 7.2, we can obtain,

$$\frac{y_{PP} - b'}{k'} = \frac{y_{GL} - b}{k} \quad (7.5)$$

where y_{PP} is the y coordinate of the picture plane and y_{GL} is the y coordinate of the ground line. Eq. (7.4) combining with Eq. (7.5) provides us with the location of L_1' . To get the two end points of L_1' , we need to use Property 7.1. Assuming that PP_L is the projection of L_1 on the picture plane, and its two end points are $PP_{L0}(x_{P0}, y_{PL})$ and $PP_{L1}(x_{P1}, y_{PL})$; $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$ are the two end points of L_1' ; k_0 and k_1 are the slopes of SP- PP_{L0} and SP- PP_{L1} . Based on the Property 7.1, we have:

$$x_0 = \frac{k_0 \cdot x_{SP} + b' - y_{SP}}{k_0 - k'}, \quad y_0 = k' x_0 + b' \quad (7.6)$$

$$x_1 = \frac{k_1 \cdot x_{SP} + b' - y_{SP}}{k_1 - k'}, \quad y_1 = k' x_1 + b' \quad (7.7)$$

We can obtain the two end points of L_1' from Eq. (7.6) and (7.7). All the lines located on the bottom plane can be redrawn by repeating this process, thus producing the plan view. Now, to obtain the 3D object, we still lack height information of each vertex.

7.5 Obtaining the Height of Each Vertex

To get the height of a vertex, we need to project the vertex onto a vertical true height line (THL) which serves as the reference for vertical height measurement. We know that all vertical measurements are taken from the ground line [76], and in a two-point perspective, the true height occurs only at the intersection between the picture plane and the plan view of the perspective drawing [78]. In the plan view of an object, there must exist a point touching the picture plane due to our previous assumption that every object touches the picture plane in 3D. So, to get the true height line, we only need to drop this point down vertically to the ground line. We have made the assumption that the vertex of a perspective drawing touching the ground line is GP, therefore, the vertical line, starting from the intersection of the picture plane and the plan view, will touch the Ground line at GP. Thus, the vertical line going through GP is the THL in this paper. Unfortunately, not every vertex can be projected on to the THL directly. Thus, we first project a vertex on to the left plane which goes through GP-LVP and THL (see Fig. 7.5). The projection direction for the left projection is perpendicular to the left plane. To do the left projection, we do not project every original edge onto the left plane directly. Since we have projected all the vertices onto the bottom plane, here, we first construct a virtual line e that connects an original vertex with its projection on the bottom plane, and e is a V_L . Therefore, the projection of the original vertex on the left plane can be obtained if e' , which is the projection of e on the left plane, is known. To perform this projection, we introduce two rules:

Rule 7.4. A V_L 's projection on the left plane still is a V_L .

Because a V_L is parallel to the left plane, its projection on the left plane remains unchanged.

Rule 7.5. The line joining a vertex and its projection on the left plane is a R_L .

The line joining a vertex and its projection on the left plane must be perpendicular to the left plane. Each line perpendicular to the left plane must converge to the right vanishing point, therefore, the line is a R_L .

Once all the vertices of the drawing are projected onto the bottom plane, to project an original vertex onto the left plane, we need only two steps,

Step 1: Let the projection of P on the bottom plane be P_B ; project P_B onto the left plane.

A point in space could be determined by two intersecting lines. We know that the bottom plane and the left plane are perpendicular to each other, and their intersection is GP-LVP, as depicted in Fig. 7.5. Therefore, a vertex on the bottom plane will be projected on to the line GP-LVP. We also know that a vertex and its projection on the left plane lie in the same R_L according to Rule 7.5. Thus, for a vertex P_B on the bottom plane, its projection P_{BL} on the left plane is the intersection of GP-LVP and P_B -RVP.

Step 2: Let the projection of P_B on the left plane be P_{BL} , then we create a vertical line L through P_{BL} , the intersection of RVP- P and L will be the projection of the vertex P on the left plane.

We know that P and P_B lie on the same V_L , so their projection on the left plane also lie in the same V_L , based on Rule 7.4. There is another condition that P and its projection on the

We take the line drawing in Fig. 7.5 as an example, the left plane containing THL and GP-LVP is marked in bold and all the lines on the bottom plane are marked in bold dot. For instance, the projection of V_5 (x_5, y_5) on the bottom plane is V_5' (x_5', y_5'), the projection of V_5' on the left plane is V_{5L}' (x_{5L}', y_{5L}'), and the projection of V_5 on the left plane is V_{5L} (x_{5L}, y_{5L}). Since V_{5L}' is the intersection of RVP- V_5' and LVP-GP, we have

We know that V_{5L} is the intersection of $RVP-V_5$ and a V_L going through V_{5L}' , so we have,

170

$$\frac{y_{5L} - y_{RVP}}{x_{5L} - x_{RVP}} = \frac{y_5 - y_{RVP}}{x_5 - x_{RVP}} \quad (7.9)$$

From Eq. (7.8), we can obtain the coordinate of V_{5L}' , and then the coordinate of V_{5L} from Eq. (7.9). All the original vertices of the line drawing can be projected onto the left plane using this process.

The next step is to project all the vertices on the left plane onto the true height line. To do this projection, we use the principle that a vertex and its projection on the true height line are on the same L_L , as depicted in Fig. 7.6, where all projected lines on the left plane are marked with bold black lines. We need to project every vertex on the projected left plane onto the true height line. Take $V_1(x_1, y_1)$ as an example, assume its projected point on the THL is $V_1'(x_1', y_1')$, according to the principle, we have,

$$x_1' = x_{GP}$$

$$\frac{y_1 - y_1'}{x_1 - x_1'} = \frac{y_1 - y_{LVP}}{x_1 - x_{LVP}} \quad (7.10)$$

where x_{GP} is the x coordinate of the GP. We can determine the y_1' as well as ground height of V_1 from Eq. (7.10). Similarly, we can obtain the ground heights of all the vertices.

Fig. 7.6. Illustration of obtaining the height of vertices.

7.6 Recovering the 3D Object

Having the plan view and the height of every vertex in hand, the vertices, and thus the 3D object are completely determined.

Assuming that the input 2D drawings are accurate two-point perspective and the two vanishing points are given, our algorithm for 3D reconstruction can be summarized as:

Algorithm 7.1: 3D object reconstruction from two-point perspective line drawing.

- 1) *Read input 2D drawings and find faces of the object*
- 2) *Construct horizon, ground line and picture plane based on given information*
- 3) *Estimate the position of station point according to the constraints and assumptions*
- 4) *Construct the bottom plane*
- 5) *Project each line segment onto the bottom plane*
- 6) *Create the plan view of each projected line segments on the picture plane*
- 7) *Construct the left plane and true height line*
- 8) *Project each line segment onto the left plane*
- 9) *Project each projected vertex onto the true height line to get its height*
- 10) *Combine the plan view and height information to recover the 3D object*

7.7 Results

Fig. 7.7 shows some 3D objects reconstructed by our algorithm. For each object, the leftmost one is the 2D line drawing of the object in perspective view; the rightmost two are reconstructed 3D objects in two different views in parallel projection. The line drawings are perspective projections from a CAD system and hence the inputs are accurate.

Fig. 7.7. Objects recovered from accurate perspective drawing.

We have also applied our method on photographic images in two-point perspective. As the same pre-processing method we introduced in Chapter 5 to handle photographic images, we manually trace lines over the edges of the objects in the photographs, giving the desired perspective line drawings. Then, we use our method to reconstruct the 3D model of the drawing. Textures are extracted from the faces of the objects in the photograph and then mapped into the corresponding faces of the reconstructed 3D object. Fig. 7.8 shows some results.

Fig. 7.8. 3D models recovered from single image with lines drawn on the edges of desired objects.

The reconstruction in this chapter is subject to scale. Here, we compare three objects' length ratios we obtained from the photos with the true values. As we can see from Table 7.1, the results we obtained are close to the true values.

Table 7.1 Object 1 and object 2 are the black and orange box respectively in Fig. 7.8a.

Length ratio	Computed results	True values
Object 1	1 : 1.107 : 1	0.976 : 1 : 1
Object 2	0.971 : 0.616 : 1	0.980 : 0.591 : 1
Object 3	1.001 : 0.709 : 1	0.994 : 0.711 : 1

Theoretically, our approach can return accurate results for accurate inputs. The differences between our results and the true values arise from three types of errors. One is

that the photographs may not be exactly in two-point perspective. Another is in the estimation of the station point. The first error comes from the photographer and it can be avoided. But the second error cannot be avoided at present, because there is no information to determine the exact projected position of the station point on the plane picture. In this chapter, we used the image center, most likely to be considered as the projection of the station point on the picture plane; it is the best estimation in most cases, but still may not be the exact solution. The third source of error is the manual insertion of the edges on a pixilated image. However accurately one can trace the edges, they can at most be accurate to three significant figures, which is roughly the order of magnitude of the number of pixels across a screen. This is a lot less than the accuracy achievable in numerical computations.

7.8 Discussion

Since this method recovers 3D objects from two orthogonal views, there exists the conformance issue between these two views, that is, whether the 3D object recovered from these two views will match the original object depicted in the 2D perspective drawing. In a perspective drawing, in terms of given plan view of the drawing, there exists only one true height which match the plan view. This true height occurs only at the intersection between the plan view and the picture plane [81].

The distance between an object and the picture plane can be a variable because, for a perspective drawing, we cannot predict how far it is away from the viewer. If the object does not touch the picture plane in 3D, there is no intersection between its plan view and the picture plane. In this case, we need to extend one edge of the plan view to touch the

picture plane, and then to get the true height line. In this chapter, we assume that the object touches the picture plane in 3D, and there exists an intersection between the plan view of the object and the picture plane. The line that drops along the intersection down vertically to the ground line is the only true height line. The height of each vertex from this true height line is matched with the given plan view.

7.9 Chapter Summary

In this chapter, a novel method to obtain the 3D object depicted in a 2D line drawing of the object in perspective view is presented, and it is implemented by reversing the standard perspective drawing process, which constructs the two-point perspective drawing of an object from its two orthogonal views, so, the major part of the method is to derive the plan view and height of each vertex of an object from its two-point perspective drawing. After obtaining the plan view and height of each vertex on the plan view, the 3D object is constructed easily by picking out the three coordinate values of the vertices.

The method is computationally efficient, since only the relationship between lines and vanishing points was considered. It has another advantage: face finding is only required in cases when a line cannot be projected directly onto the bottom plane, which means that in many cases, faces are not required. Most existing methods including the one we introduced in chapter 5 that recover 3D objects from perspective drawings need face information, which should be avoided at best.

The method has a limitation: some drawings cannot be recovered fully because some lines are “isolated”. Overcoming this limitation will enable us to recover polyhedral objects in general; this is an issue for future work. Another issue for the future is in

multiple images. The method can reconstruct an object completely if the drawing is complete, that is, contains every edge in the object. But unfortunately, this is not always true, especially in single photographic images. So multiple images are required, and this then requires the merging of reconstructed results, which is another problem for the future.

CHAPTER 8

Conclusion

8.1 Summary and Contributions

This thesis has presented work on 3D object reconstruction from single line drawings in parallel and perspective projections, and the preprocessing of the drawings before the 3D reconstruction. The preprocessing includes decomposition of complex drawings into multiple simpler drawings of individual manifolds and identification of the faces of the object in the drawing. Three methods for recovering 3D object were presented: cubic corner method, hybrid method and the reverse engineering of perspective drawings.

Chapter 3 presented algorithms to decompose complex line drawings which depict connected 3D objects into multiple simpler drawings of individual manifolds without face finding. The decomposition process has three stages: decomposition at non-manifold vertices, along non-manifold edges and internal faces. The first two decompositions are relatively simple; once the potential non-manifold vertices and edges are found, the decomposition can be performed directly. Thus the major task is decomposition across internal faces which has two steps: basic decomposition and repair for incomplete parts. The decomposition is performed without requiring face identification which is computationally expensive, and is made at existing vertices and edges, without introducing any new ones. After decomposition, the time for face finding is much reduced and this, in turn, greatly improves the process for 3D reconstruction, which is the ultimate goal.

The new face identification algorithm, introduced in Chapter 4, consists of two stages: decomposition and face forming. It first decomposes a line drawing into chains of edges, each belonging to only one face of the object, determined using simple rules. Most of these chains are not closed. That is because the simple rules normally cannot pick out all the edges belonging to one face. The faces are then formed from the chains by finding their missing edges based on some more rules. This new face finding method is totally different from existing ones and it delivers both the real and the internal faces, whereas existing algorithms deliver only the real faces. In addition, it is highly efficient. Comparison with the latest method shows that it is faster often by tens of times and, in some cases, hundreds of times.

Chapter 5 presented a direct method, the cubic corner method, to recover the geometry of the 3D polyhedron depicted in a single parallel or perspective projection. In parallel projection, it uses two sets of information, the list of faces in the object, obtained automatically from the drawing, and a user-identified cubic corner, to compute for the coordinates of the vertices in the drawing and thus establish the 3D geometry of the whole polyhedron. In perspective projection, before proceeding to compute the coordinates of the vertices in the drawing, focal length need to be found based on vanishing point information to establish the linear relationship between a vertex in the drawing and its corresponding point on the object. The algorithm exploits the topological structure of the polyhedron, implicit in the connectivities between the faces, resulting in a complexity linear in the number of faces. The method is extended to objects with no cubic corners as well. The 3D reconstruction algorithm is simple and direct for the simple fact that given the geometric and topological constraints that exist in a 2D drawing, a 3D

object which is a simple manifold is completely defined once the coordinates of the first four points at a corner, and hence the three faces that contain them, are known. It is very efficient because it has to visit each face at most once.

In Chapter 6, we introduced a hybrid method to reconstruct 3D polyhedral objects from 2D line drawings by combining two known methods, cubic corner method and optimization-based method and presented comprehensive test results comparing the three methods. The hybrid method starts with the cubic corner method, and uses its output as the initial guess of the optimization process. Tests are conducted for each method using drawings of varying degrees of accuracy. The results show that it works well for inaccurate drawings. It overcomes the deficiencies of both the cubic corner method and the optimization-based method to produce a result much better than that attainable by either method alone.

Chapter 7 presents a novel algorithm to reconstruct the 3D object depicted in an accurate 2D line drawing in a 2-point perspective view. It reverses the technique used to draw perspective views of 3D objects taught in drawing classes, which creates a perspective view from two orthogonal planar projections. Our method first recreates the plan view of the object, which gives us the x and y coordinates of the vertices, and then obtains the height (the z -coordinate) of each vertex separately by recreating the side projection. The plan view and the height together give the x , y , z coordinates of the vertices, which define the object in 3D. The method is able to reconstruct 3D object without computing for the focal length which is indispensable for existing methods. It is computationally efficient, since only the relationship between lines and vanishing points was considered. It has another advantage: face information is only required in cases when a line cannot be

projected directly onto the bottom plane, which means that in many cases, faces are not required.

8.2 Conclusion and Future Work

As we mentioned in last section, the work in this thesis makes significant contributions in the area of human-computer communication. However, some algorithms have deficiencies.

- As we mentioned earlier, in some cases, the decomposition algorithm results in no decomposition. This leads additional cost to the overall performance of face finding. Although the additional cost is very small which would not unduly affect the face finding process, avoiding it is still necessary. An algorithm for deciding beforehand whether a drawing can be decomposed into simpler manifolds deserves our attention in the future.
- The new face finding method of Chapter 4 works efficiently for most objects except for some special cases containing certain types of hole which adjoin internal faces. Although the problem is overcome by working through the permutation of sub-loop groupings partitioned by internal faces, this permutation gives rise to a factorial complexity over the number of groups, which is undesirable. In addition, the method needs the user to state the genus of the object, which should be avoided. In the future, we will be working to seek an alternative, particularly in the ability to identify holes in a drawing directly.
- The 3D reconstruction algorithms we have introduced apply to polyhedral objects only. Real life objects contain planar and curved faces, and it is therefore necessary to

extend the algorithm to deal with drawings of such objects, which means dealing with drawings containing both straight and curved edges. Some work has been done within our research team in this area [82]. The author has also contributed to solving the problem [83], but this work is still rather tentative and so has not been included in this thesis. But the idea developed in [83] is promising and can be taken further.

- The reverse engineering method in Chapter 7 recovers 3D objects in 2-point perspective view by obtaining the plan view and side view of the objects. It is very useful for architectural drawings and objects in photographs. Face finding is not necessary for the method, it is therefore very efficient. But it has a limitation: some drawings cannot be recovered fully because some lines are “isolated”. Overcoming this limitation is an important issue in our future work.
- The cubic corner method can recover 3D object exactly, and this leads to the prospect of recovering objects in photographs accurately, if we can extract edges accurately from photographs. The reconstruction of objects from photographs accurately has useful applications, such as the reconstruction of the geometry of existing engineering components, which is done using expensive 3D laser scanners currently. This issue, including accurate automatic detection of edges from photographs, is part of our future work.
- One issue common to all the reconstruction methods is that of multiple images. The methods can reconstruct an object completely if the drawing is complete, that is, contains every edge in the object. But unfortunately, this is not always possible. For example, an object, such as a building, usually requires multiple photographs to cover it completely. This then requires merging of the multiple views, either before or after

reconstruction. This is an interesting problem for the future, particularly for reconstruction from inaccurate inputs, as the consistency of the results becomes an issue here.

List of Publications

1. YT Lee, **F Fang**, “3D reconstruction of polyhedral objects from single parallel projections using cubic corner,” *Computer Aided Design*, Vol. 43, No. 8, pp. 1025-1034, 2011.
2. YT Lee, **F Fang**, “A new hybrid method for 3D object recovery from 2D drawings and its validation against the cubic corner method and the optimisation-based method,” *Computer Aided Design*, Vol. 44, No. 11, pp. 1090-1102, 2012.
3. **F Fang**, YT Lee, “3D reconstruction of polyhedral objects from single perspective projections using cubic corner,” *3D Research*, Vol. 3, No. 2, pp. 1-8, 2012.
4. **F Fang**, YT Lee, “3D reconstruction from drawings with straight and curved edges,” *SIGGRAPH Asia-Technical Brief*, DOI:10.1145/2542355.2542357, 2013.
5. MC Leong, YT Lee, **F Fang**, “A Search-and-Validate Method for Face Identification from Single Line Drawings,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 11, pp. 2576-2579, 2013.
6. **F Fang**, YT Lee, “Efficient decomposition of line drawings of connected manifolds without face identification,” *Computer Aided Design*, Vol. 51, No. 6, pp. 18-30, 2014.
7. **F Fang**, YT Lee, MC Leong, “Identification of faces in line drawings by edge decomposition,” *Pattern and Recognition*, under review.

References

- [1] H. Lipson and M. Shpitalni, "Optimization-based reconstruction of a 3D object from a single freehand line drawing," *Computer Aided Design*, Vol. 28, No. 8, pp. 651-663, 1996.
- [2] J. Liu, L. Cao, Z. Li, and X. Tang, "Plane-based optimization for 3D object reconstruction from single line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 30, No. 2, pp. 315-327, 2008.
- [3] Y. Chen, J. Liu, and X. Tang, "A divide-and-conquer approach to 3D object reconstruction from line drawings," *IEEE International Conference on Computer Vision 2007*, pp. 1-8, 2007.
- [4] T. Xue, J. Liu, and X. Tang, "Object cut: Complex 3D object reconstruction through line drawing separation," *IEEE Conference on Computer Vision and Pattern Recognition 2010*, pp. 1149-1156, 2010.
- [5] J. R. Munkres, "Topology," Prentice Hall, 2000.
- [6] M. C. Leong, Y. T. Lee, and F. Fang, "A Search-and-Validate Method for Face Identification from Single Line Drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 11, pp. 2576-2591, 2013.
- [7] D. A. Huffman, B. Meltzer, and D. Michie, "Impossible objects as nonsense sentences " *Machine Intelligence*, Vol. 6, pp. 295-323, 1971.
- [8] M. B. Clowes, "On seeing things," *Artificial Intelligence*, Vol. 2, No. 2, pp. 79-116, 1971.
- [9] A. Guzman, "Decomposition of a visual scene into three-dimensional bodies," *AFIPS'68 Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pp. 291-304, 1968.
- [10] M. C. Cooper, "Line Drawing Interpretation," *Springer*, 2008.
- [11] R. R. Martin, H. Suzuki, and P. A. C. Varley, "Labeling engineering line drawings using depth reasoning," *Journal of Computing and Information Science in Engineering*, Vol. 5, No. 6, pp. 158-167, 2005.
- [12] P. A. C. Varley, "Automatic Creation of Boundary-Representation Models from Single Line Drawings," *Ph.D Thesis, Cardiff University*, 2003.
- [13] T. Kanade, "Recovery of the three-dimensional shape of an object from a single view," *Artificial Intelligence*, Vol. 17, No. 5, pp. 409-460, 1981.

- [14] K. Sugihara, "A necessary and sufficient condition for a picture to represent a polyhedral scene," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, No. 5, pp. 578-586, 1984.
- [15] M. C. Cooper, "Wireframe projections: Physical realisability of curved objects and unambiguous reconstruction of simple polyhedra," *International Journal of Computer Vision*, Vol. 64, No. 1, pp. 69-88, 2005.
- [16] T. Marill, "Emulating the human interpretation of line-drawings as three-dimensional objects," *International Journal of Computer Vision*, Vol. 6, No. 2, pp. 147-161, 1991.
- [17] E. Brown and P. Wang, "3D object recovery from 2D images: A new approach," *Proceedings of the SPIE on Robotics and Computer Vision*, pp. 138-145, 1996.
- [18] K. Shoji, K. Kato, and F. Toyama, "3D interpretation of single line drawings based on entropy minimization principle," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, pp. 1190-1195, 2001.
- [19] 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*, Vol. 9, No. 2, pp. 113-136, 1992.
- [20] A. Piquet, R. R. Martin, and P. Company, "Using skewed mirror symmetry for optimization-based 3D line drawing recognition," *Proceedings of the Fifth IAPR International Workshop Graphics Recognition*, pp. 182-193, 2003.
- [21] Y. Wang, Y. Chen, J. Liu, and X. Tang, "3D reconstruction of curved objects from single 2D line drawings," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1834-1841, 2009.
- [22] J. Chu, M. Gao, G. Zhang, and R. Feng, "Line-based optimization for 3D object reconstruction from single line drawings," *Applied Mechanics and Materials*, Vol. 20-23, No. 1, pp. 910-915, 2010.
- [23] C. K. Fong and W. K. Cham, "3D object reconstruction from single perspective distorted line drawing image using vanishing points," *Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 53-56, 2005.
- [24] D. Jelinek and C. J. Taylor, "Reconstruction of linearly parameterized models from single images with a camera of unknown focal length," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, No. 7, pp. 767-773, 2001.
- [25] Z. Li, J. Liu, and X. Tang, "Shape from regularities for interactive 3D reconstruction of piecewise planar objects from single images," *Proceedings of the 14th annual ACM International Conference on Multimedia*, pp. 85-88, 2006.

- [26] Z. Li, J. Liu, and X. Tang, "A closed-form solution to 3D reconstruction of piecewise planar objects from single images," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-6, 2007.
- [27] A. Criminisi, I. Reid, and A. Zisserman, "A plane measuring device," *Image and Vision Computing*, Vol. 17, No. 8, pp. 625-634, 1999.
- [28] A. Criminisi, I. Reid, and A. Zisserman, "Single view metrology," *International Journal of Computer Vision*, Vol. 40, No. 2, pp. 123-148, 2000.
- [29] A. Criminisi, "Accurate visual metrology from single and multiple uncalibrated images," *Springer*, 2001.
- [30] J. M. Loomis, "Looking down is looking up," *Nature*, Vol. 414, No. 11, pp. 155-156, 2001.
- [31] B. K. P. Horn and M. J. Brooks, "Shape from shading," *The MIT Press*, 1989.
- [32] R. Zhang, P. S. Tsai, J. E. Cryer, and M. Shah, "Shape from shading :A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21, No. 8, pp. 690-706, 1999.
- [33] A. Blake and C. Marinos, "Shape from texture. Estimation, isotropy and moments," *Artificial Intelligence*, Vol. 45, No. 8, pp. 323-380, 1990.
- [34] A. Loh and R. Hartley, "Shape from non-homogeneous, non-stationary, anisotropic, perspective texture," *Proceedings of the British Machine Vision Conference 2005*, pp. 1-10, 2005.
- [35] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," *Proceedings of Advances in Neural Information Processing Systems 18*, pp. 1161-1168, 2005.
- [36] A. Saxena, M. Sun, and A. Y. Ng, "Make3D: Learning 3D scene structure from a single still image," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, No. 5, pp. 824-840, 2009.
- [37] M. Masry, D. J. Kang, I. Susilo, and H. Lipson, "A freehand sketching interface for progressive construction and analysis of 3D objects," *Journal of Computers and Graphics*, Vol. 29, No. 5, pp. 563-575, 2005.
- [38] L. Cao, J. Liu, and X. Tang, "3D object retrieval using 2D line drawing and graph based relevance feedback," *Proceedings of ACM International Conference on Multimedia*, pp. 105-108, 2006.
- [39] Y. Wang, Y. Chen, J. Liu, and X. Tang, "3D reconstruction of curved objects from single 2D line drawings," *IEEE Conference on Computer Vision and Pattern Recognition 2009*, pp. 1834-1841, 2009.

- [40] Y. Liu and Y. T. Lee, "Optimization-based 3D reconstruction of freeform shapes from 2D line drawings," *The 14th International Conference on Computer Graphics and Artificial Intelligence*, 2012.
- [41] M. Oswald, R., E. Toppe, and D. Cremers, "Fast and globally optimal single view reconstruction of curved objects," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 534-541, 2012.
- [42] C. Shao, A. Bousseau, A. Sheffer, and K. Singh, "CrossShade: shading concept sketches using cross-section curves," *ACM Transactions on Graphics, SIGGRAPH*, Vol. 31, No. 4, 2012.
- [43] Y. Sun and Y. T. Lee, "Topological analysis of a single line drawing for 3D shape recovery," *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pp. 167-172, 2004.
- [44] J. Liu, Y. Chen, and X. Tang, "Decomposition of complex line drawings with hidden lines for 3D planar-faced manifold object reconstruction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, No. 1, pp. 3-15, 2011.
- [45] J. Liu and Y. T. Lee, "A graph-based method for face identification from a single 2D line drawing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, No. 10, pp. 1106-1119, 2001.
- [46] J. Liu and X. Tang, "Evolutionary search for faces from line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27, No. 6, pp. 861-872, 2005.
- [47] G. Markowsky and M. A. Wesley, "Fleshing out wire frames," *IBM Journal of Research and Development*, Vol. 24, No. 5, pp. 582-597, 1980.
- [48] P. M. Hanrahan, "Creating volume models from edge-vertex graphs," *ACM SIGGRAPH Computer Graphics*, Vol. 16, No. 3, pp. 77-84, 1982.
- [49] R. D. Dutton and R. C. Brigham, "Efficiently identifying the faces of a solid," *Computers and Graphics*, Vol. 7, No. 2, pp. 143-147, 1983.
- [50] M. A. Ganter and J. J. Uicker Jr, "From Wire-Frame to Solid-Geometric: Automated Conversion of Data Representations," *Computers in Mechanical Engineering*, Vol. 2, pp. 40-45, 1983.
- [51] S. Courter, M. and J. Brewer, A., "Automated Conversion of Curvilinear Wire-Frame Models to Surface Boundary Models: A topological Approach," *SIGGRAPH' 86 Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 20, pp. 171-178, 1986.

- [52] S. Bagali and J. Waggenspack, W. N., "A Shortest Path Approach to Wireframe to Solid Model Conversion," *Proceedings of the Third Symposium on Solid Modeling and Applications*, pp. 339-349, 1995.
- [53] M. Shpitalni and H. Lipson, "Identification of faces in a 2d line drawing projection of a wireframe object," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 10, pp. 1000-1012, 1996.
- [54] J. Liu, Y. T. Lee, and W. K. Cham, "Identifying faces in a 2D line drawing representing a manifold object," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 12, pp. 1579-1593, 2002.
- [55] P. A. C. Varley and P. P. Company, "A new algorithm for finding faces in wireframes," *Computer Aided Design*, Vol. 42, No. 4, pp. 279-309, 2010.
- [56] X. D. Du and L. N. Yi, "Identifying faces based on three-dimensional geometrical properties from a single 2D line drawing," *International Conference on Graphic and Image Processing*, 2011.
- [57] P. Company, M. Contero, J. Conesa, and A. Piquer, "An optimisation-based reconstruction engine for 3D modelling by sketching," *Computers and Graphics (Pergamon)*, Vol. 28, No. 6, pp. 955-979, 2004.
- [58] H. Li, Q. Wang, L. Zhao, Y. Chen, and L. Huang, "nD object representation and detection from single 2D line drawing," *Lecture Notes in Computer Science*, Vol. 3519, No. 10, pp. 363-382, 2005.
- [59] D. Perkins, "Cubic Corners," *Quarterly Progress Report 89, MIT Research Laboratory of Electronics*, pp. 207-214, 1968.
- [60] P. A. C. Varley, R. R. Martin, and H. Suzuki, "Frontal Geometry from Sketches of Engineering Objects: Is Line Labelling Necessary?" *Computer Aided Design*, Vol. 37, No. 12, pp. 1285-1307, 2005.
- [61] A. Andre, S. Saito, and M. Nakajima, "CrossSketch: Freeform surface modeling with details," *SBIM' 07 Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modelling*, pp. 45-52, 2007.
- [62] A. Andre, S. Saito, and M. Nakajima, "Single-view sketch based surface modeling," *IEICE Transactions on Information and Systems*, Vol. E92-D, No. 6, pp. 1304-1311, 2009.
- [63] Y. S. Suh, "Reconstructing polyhedral swept volumes from a single-view sketch," *IEEE International Conference on Information Reuse and Integration 2006*, pp. 585-588, 2006.

- [64] M. J. Magee and J. K. Aggarwal, "Determining vanishing points from perspective images," *Computer Vision, Graphics and Image Processing*, Vol. 26, No. 2, pp. 256-267, 1984.
- [65] C. Rother, "A new approach to vanishing point detection in architectural environments," *Image and Vision Computing*, Vol. 20, No. 9, pp. 647-655, 2002.
- [66] B. Caprile and V. Torre, "Using vanishing points for camera calibration," *International Journal of Computer Vision*, Vol. 4, No. 2, pp. 127-139, 1990.
- [67] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2004.
- [68] D. Marr, Ed., *Vision. A computational investigation into the human representation and processing of visual information*. W. H. Freeman and Company, 1982.
- [69] F. C. Langbein, A. D. Marshall, and R. R. Martin, "Choosing consistent constraints for beautification of reverse engineered geometric models," *Computer Aided Design*, Vol. 36, No. 3, pp. 261-278, 2004.
- [70] H. L. Zou and Y. T. Lee, "Constraint-based beautification and dimensioning of 3D polyhedral models reconstructed from 2D sketches," *Computer Aided Design*, Vol. 39, No. 11, pp. 1025-1036, 2007.
- [71] P. A. C. Varley, "Automatic creation of boundary-representation models from single line drawings," *Ph. D. thesis. University of Wales*, 2003.
- [72] Y. T. Lee and F. Fang, "3D reconstruction of polyhedral objects from single parallel projections using cubic corner," *Computer Aided Design*, Vol. 43, No. 8, pp. 1025-1034, 2011.
- [73] P. Company, A. Piquer, M. Contero, and F. Naya, "A survey on geometrical reconstruction as a core technology to sketch-based modeling," *Computers and Graphics (Pergamon)*, Vol. 29, No. 6, pp. 892-904, 2005.
- [74] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge, "Sketch-based modeling: A survey," *Computers and Graphics (Pergamon)*, Vol. 33, No. 1, pp. 85-103, 2009.
- [75] D. Lamb and A. Bandopadhyay, "Interpreting a 3D object from a rough 2D line drawing," *IEEE Conference on Visualization 1990*, pp. 59-66, 1990.
- [76] A. W. Griffin and V. Alvarez-Brunicardi, Eds., *Introduction to Architectural Presentation Graphics*. Prentice Hall, 1998.
- [77] J. D'amelio, Ed., *Perspective Drawing Handbook*. Dover Publications, 2004.
- [78] R. P. Gordon, Ed., *Perspective Drawing: A Designer's Method*. Fairchild Books, Inc., 2008.

- [79] C. W. Tyler, "A horopter for two-point perspective," *Proceedings of the International Society for Optical Engineering 2005*, pp. 306-315, 2005.
- [80] A. Griffin, W. and V. Alvarez-Brunicardi, *Introduction to Architectural Presentation Graphics*. Prentice Hall, 1998.
- [81] R. P. Gordon,, *Perspective Drawing: A Designer's Method*. Fairchild Books, Inc., 2008.
- [82] Y. Liu, "Recovery of 3D shapes from 2D sketches," *Ph. D. Thesis. Nanyang Technological University*, 2013.
- [83] F. Fang and Y. T. Lee, "3D reconstruction from drawings with straight and curved edges," *Proceedings of Siggraph Asia 2013 Technical Briefs*, Hong Kong, 2013.