Constraint-based beautification of 3D polyhedral objects constructed from 2D line drawings

Zou, Hong Liu

2008

Zou, H. L. (2008). Constraint-based beautification of 3D polyhedral objects constructed from 2D line drawings. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/41577>

<https://doi.org/10.32657/10356/41577>

Abstract

Designers commonly use 2D sketches to express their design contents. It would be desirable to have a system that can automatically reconstruct a 3D model from a 2D line drawing. This thesis presents a constraint-based beautification method for 3D polyhedral objects constructed from 2D line drawings.

Due to the inaccuracy inherent in a sketch and the nature of the reconstruction method, the 3D model reconstructed is rough and not quite ready. It is necessary to have an efficient method that can adjust a rough 3D model to an accurate one, as required by the designer. This thesis investigates the processes involved in "beautifying" the object and adds to the capability of the existing system.

In this work, the adjustment is made by first identifying a set of constraints from the rough model. An efficient graph-based method, which exploits the properties of a 3D rigid subgraph, is then employed to remove certain redundant or inconsistent constraints of the 3D model. This is followed by a more costly numerical method to detect and remove the redundant constraints. The result is a sufficient and consistent set of constraints. The 3D model is then adjusted to produce the beautified 3D model, together with its dimensions.

Zou Hongliu

Symmetry is an important constraint. It reduces the size of the object data to only one portion of the symmetry, and thus the computational cost of beautification. This thesis introduces a new method to detect skewed symmetry from a drawing, including both skewed mirror symmetry and skewed rotational symmetry. The graph of the drawing is first analysed for topological symmetry, and then geometric symmetry. Topological symmetry is established by checking whether the components about the plane or axis of symmetry are topologically equivalent. Then these topologically symmetric components are checked for geometric symmetry.

The algorithms have been implemented in a software for the reconstruction of 3D models from sketches. The results of our experiments show that the algorithms work well, with adjustable fault tolerance.

School of Mechanical & Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2008

Abstract

Designers commonly use 2D sketches to express their design concepts. It would be useful to directly generate the 3D model from such a sketch, and transmit this model to a CAD system for further processing. This work was proposed in NTU and a system has been developed to reconstruct a freehand design sketch of a polyhedral object into the object in 3D.

Due to the inaccuracy inherent in a sketch and the nature of the reconstruction method, the 3D model recovered is rough and not quite ready. It is necessary to have an efficient method that can adjust a rough 3D model to an accurate one, as required by the designer. This thesis investigates the processes involved in “beautifying” the object and adds to the capability of the existing system.

In this work, the adjustment is made by first identifying a set of constraints from the rough model. An efficient graph-based method, which exploits the properties of a 3D rigid subgraph, is then employed to remove certain redundant or inconsistent constraints of the 3D model. This is followed by a more costly numerical method to detect and remove the redundant or inconsistent constraints. The result is a sufficient and consistent set of constraints, which is then solved to produce the beautified 3D model, together with its dimensions.

Symmetry is an important constraint; it reduces the size of the object data to only one portion of the symmetry, and thus the computational cost of beautification. This thesis introduces a new method to detect skewed symmetry from a drawing, including both skewed mirror symmetry and skewed rotational symmetry. The graph of the drawing is first analysed for topological symmetry and then geometric symmetry. Topological symmetry is established by checking that separate components about the plane or axis of symmetry are topologically equivalent. Then these topologically symmetric components are checked for geometric symmetry.

The algorithms have been implemented in our software for the reconstruction of 3D models from sketches. The results of our experiments show that the algorithms work well, with adjustable fault tolerance.

This work has focused attention on detecting redundant and inconsistent constraints and selecting constraints to form a well-constrained system. A set of simple priorities rules is defined to determine which constraint is checked for redundancy first. However, this set of simple rules does not always lead to the desired dimensioning of a model. The problem of how to refine the priorities of the detected constraints merits further investigation.

Clearly, polyhedral objects are but a very small fraction of real world objects. Another important future work is in the extension of the capabilities to cover curved objects.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my supervisor, Dr. Lee Yong Tsui, for his patient supervision and kindness in all aspects of my research work, and for many helpful comments on the draft of this thesis.

I am indebted to NTU for awarding me a research scholarship.

I thank the staff and students of the Advanced Design and Modelling Laboratory for their help and support and for creating a pleasant and friendly working environment. I would like to thank Dr. Khoo Li Pheng, Dr. Thimm Georg, Dr. Ma Yongsheng and Dr. Tian Feng for reading and commenting on my First Year Report. I wish to thank my colleagues, Sun Yuan, Ramaiyan Murali, Pei Shuen Cheo, Lu Baifang, Yuan Yu, Fan Zhaowei, Guan Yunqing, Chandrasekaran Indhumathi, Lin Jietong and Wang Jiabing for many helpful discussions about my project and programming, as well as my English writing.

I would like to thank Dr. Christophe Jermann of the University of Nantes for stimulating discussions on the degree of rigidity computation. Many thanks to SIAM for providing us LAPACK and to Prof. Sven Hammarling from the Numerical Algorithms Group for the helpful discussion on using LAPACK. I also thank Prof. Robert Joan-Arinyo of Universitat Politècnica de Catalunya and Prof. Rafael Bidarra of Delft University of Technology for valuable discussions and encouragement about our research on constraint solving during the CAD05 conference in Thailand.

I appreciate all the help, encouragement and advice from my friends, and cherish the good times we have had together. Special thanks to the many developers of free

software whose systems I have used in my implementation. Without them the presented system could not have been implemented so effectively.

Last but not least, I wish to thank my parents and my brother for their understanding, encouragement and moral support.

List of Publications

- 1 Zou H.L. and Lee Y.T. Skewed mirror symmetry detection from a 2D sketch of a 3D model. In: International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, Dunedin, New Zealand, 2005.
2. Zou H.L. and Lee Y.T. Detecting minimum over-constrained subgraphs in 2D and 3D based on degree of freedom analysis. Computer-Aided Design and Applications, 2: 393-402, 2005.
3. Zou H.L. and Lee Y.T. Skewed rotational symmetry detection from a 2D line drawing of a 3D polydral object. Computer-Aided Design, 38: 1224-1232, 2006.
4. Lee Y.T. and Zou H.L. Ellipse fitting for skewed rotational symmetry. In: International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, Kuala Lumpur, Malaysia, 2006.
5. Zou H.L. and Lee Y.T. Constraint-based beautification and dimensioning of 3D polyhedral models reconstructed from 2D sketches. Computer-Aided Design, 39: 1025-1036, 2007.

Table of Contents

Abstract.....ii

Acknowledgements iii

List of Publicationsv

Table of Contentsvi

List of Figures.....xi

List of Tables.....xiv

CHAPTER 1 Introduction.....1

1.1 Background and motivation.....1

1.2 Thesis objectives and scope.....6

1.3 Organization of the thesis7

CHAPTER 2 Literature Review and Problem Statement.....8

2.1 Literature review on sketching systems8

2.1.1 Interactive construction approach8

2.1.2 Line labelling10

2.1.3 Linear system approach.....11

2.1.4 Nonlinear system approach.....13

2.1.5 A two-step approach.....14

2.2 Constraints15

2.3 Constraint solving21

2.3.1	Rule-based approach	22
2.3.2	Graph-based approach.....	23
2.3.3	Symbolic approach.....	26
2.3.4	Numerical approach	27
2.4	Problem statement.....	28
CHAPTER 3 Skewed Mirror Symmetry Detection from a 2D Sketch of a 3D Model		
		32
3.1	Introduction.....	32
3.2	Assumptions	34
3.3	Previous work	35
3.4	Topological analysis for face identification.....	40
3.5	Detecting skewed symmetry of polygonal shapes and rules for checking skewed symmetry.....	42
3.5.1	Rules for skewed symmetry in a polygon	42
3.5.2	Rules for skewed symmetry in a polyhedron	44
3.6	Detecting symmetry plane in a model.....	44
3.7	Complexity of symmetry detecting algorithm.....	51
3.8	Results	51
3.9	Conclusions and discussions.....	53

CHAPTER 4 Skewed Rotational Symmetry Detection from a 2D Line Drawing of a
3D Polyhedral Object55

4.1 Introduction.....55

4.2 Previous work56

4.3 Geometric analysis for skewed rotational symmetry in a vertex set.....57

4.3.1 Fitting a single ellipse.....60

4.3.2 Mapping of an ellipse to a circle.....66

4.3.3 Fitting an ellipse to four vertices68

4.3.4 The order of rotational symmetry in a polygon71

4.3.5 Multiple sets of rotationally symmetric vertices.....72

4.4 Topological analysis.....73

4.4.1 Topological symmetry detection in a 2D drawing of a 3D model.....73

4.4.2 Complexity of the algorithm.....81

4.5 Results82

4.6 Conclusions.....84

CHAPTER 5 Detecting Minimum Over-constrained Subgraphs in 2D and 3D Based
on Degree of Freedom Analysis.....86

5.1 Introduction.....86

5.2 Finding over-constrained subgraphs89

5.2.1 Definitions and graphical representation.....90

5.2.2	Property of rigid subgraphs	93
5.2.3	Algorithm for finding a minimum over-constrained subgraph.....	97
5.3	Complexity analysis	104
5.4	Example of application.....	104
5.4.1	A simple 2D example	104
5.4.2	A simple 3D example	106
5.4.3	Another 3D example	107
5.5	Conclusions.....	108
CHAPTER 6 Beautification and Dimensioning of the 3D Model Reconstructed from a 2D Sketch..... 110		
6.1	Introduction.....	110
6.2	Constraint system	117
6.2.1	Entities.....	117
6.2.2	Constraint equations	118
6.2.3	Detecting the constraints	120
6.3	Selecting and solving the constraints	122
6.3.1	Solving the constraints	124
6.3.2	Eliminating structurally redundant constraints	125
6.3.3	Detecting numerical redundancy	126
6.3.4	A simple example for redundancy detection	132

6.4	Changing dimensions by the user	135
6.5	Examples	136
6.6	Symmetry and beautification	142
6.7	Conclusions.....	142
CHAPTER 7 Summary and Future Work.....		144
7.1	Summary of accomplishments.....	144
7.1.1	Skewed symmetry detection	144
7.1.2	Beautification and dimensioning of a recovered model.....	145
7.2	Discussions and future work.....	146
7.3	Conclusion	151
References		153

List of Figures

Figure 1.1: Conceptual design by sketching2

Figure 1.2: Overview of system structure.....3

Figure 2.1: Faces showing skewed symmetry..... 19

Figure 2.2: Faces showing skewed orthogonality.20

Figure 2.3: Two examples of prismatic face.20

Figure 3.1: Three forms of geometric symmetry.....33

Figure 3.2: Two isomorphic graphs G1 and G2.37

Figure 3.3: A model and circuits corresponding to actual faces40

Figure 3.4: Potential faces of the depicted object41

Figure 3.5: Candidate face symmetry axes.....42

Figure 3.6: Types of edges..45

Figure 3.7: Edge pairs and face pairs.46

Figure 3.8: Symmetry polygon.48

Figure 3.9: A model and all its symmetry planes if we run the algorithm
without the rules for checking symmetry.50

Figure 3.10: Another example with the same adjacency matrix as the example in
Figure 3.951

Figure 3.11: Some examples of input sketches and their skewed symmetry planes.
.....52

Figure 3.12: An example of ambiguity..54

Figure 4.1: Examples of rotational symmetry.....56

Figure 4.2: A rotationally symmetric object and the circles containing symmetric
vertices..59

Figure 4.3: A fitted ellipse and a vertex.....65

Figure 4.4: A circle and its parallel projection.66

Figure 4.5: Three different ellipses that fit four points.....68

Figure 4.6: Polygons with interlaced subsets of vertices.....71

Figure 4.7: Rotationally symmetric objects and their symmetric face sets about
the axes shown.....75

Figure 4.8: Rotationally symmetric 3D objects and their axes of symmetry.....75

Figure 4.9: Detecting the skewed rotational symmetry beginning from a vertex.
.....76

Figure 4.10: Sequence of finding face sets78

Figure 4.11: Examples83

Figure 4.12: Two objects with axes through holes.....84

Figure 5.1: A 3D example.90

Figure 5.2: A 2D example of degree of rigidity.93

Figure 5.3: Variables in the algorithm.98

Figure 5.4: A 2D example on finding minimum over-constrained subgraph. ...106

Figure 5.5: A 3D example.106

Figure 5.6: A 3D example using a pyramid.108

Figure 6.1: Conceptual design by sketching.111

Figure 6.2: Pappas’s Theorem.....114

Figure 6.3: An example of an implicit constraint in 3D. 114

Figure 6.4: Base face of a parallel faces set..... 121

Figure 6.5: Flow chart for determining if a constraint is redundant or inconsistent.
..... 131

Figure 6.6: An example with numerical redundant constraint. 132

Figure 6.7: Test examples. 141

Figure 7.1 Mirror symmetry and rotational symmetry. 148

Figure 7.2: Examples of partial symmetry..... 148

List of Tables

Table 5.1: Freedoms of geometric entities in 3D.91

Table 5.2: Basic constraint types.92

Table 6.1: Representation of entities. 118

Table 6.2: Constraint equations..... 119

Table 6.3: Priority values of constraints, a higher value denotes a higher priority.
..... 122

Table 6.4: The Jacobian Matrix for the example of Figure 6.6..... 134

Table 6.5: Time (in seconds) for redundancy detection..... 135

Table 6.6: Computing times (seconds) for objects in Figure 6.7 137

CHAPTER 1

Introduction

1.1 Background and motivation

Emulating and explaining a human's ability to mentally recover the spatial shape of objects from the images received on the retina has been a major goal of machine vision and artificial intelligence during the past few decades [Roberts 1963; Lipson 1998]. Several experiments suggest that in many cases boundary or surface information is the principal shape cue [Barrow 1981]. Information about surfaces can be obtained from many sources such as texture gradient, stereopsis, motion parallax, shading and so on.

Traditionally designers often conceptualize 3D objects by sketching their ideas with pencil and paper and refining them into workable designs. Unfortunately, this technique is different from those for designing objects using a 3D CAD system, which does not offer a natural interface to conceptual design. Here therefore lies the fundamental function with which CAD systems do not cope well: allowing designers to express their ideas in a more natural way, i.e. through sketching. The need for operating menus, commands and other interface gadgets impedes the creative processes in a design. These interactivities place a mental load on the designer. It would be useful to have a CAD tool that accepts and understands designs in a rough and inaccurate form as in a design sketch, without requiring details. That is, get a CAD system to understand 2D sketches and return to the designer the 3D solids they

depict.

Jenkins and Martin [Jenkins 1993] pointed out that rough sketching is important in terms of flexibility and speed. Lipson [Lipson 1996] concurred, saying that sketching appeared to be important for being fast and implicit, but is inexact and requires little commitment.

Figure 1.1: Conceptual design by sketching

Figure 1.1 illustrates the problem to be tackled: take as input a freehand line drawing of a single object and produce a 3D object model from it. No artificial restrictions are to be imposed on the freehand drawing process. No user input should be required other than the sketch itself.

Figure 1.2 shows the stages involved in recovering a 3D model from a line drawing.

In this project, the task of interpreting a line-drawing is divided into five parts: 2D tidying, finding faces, solving for vertex geometry, beautification and dimensioning. The reconstruction method that has been developed so far deals only with polyhedral objects. Consequently, only polyhedral objects are considered in this thesis.

In this thesis, the term “sketch” and “line drawing” are used interchangeably. A

sketch, which is rough and may contain defects such as gaps between edges, need to be “tidied” into a proper line drawing. The inputs to the algorithms developed in this work are always the tidied version, and assume that the sketches have been tidied first.

Figure 1.2: Overview of system structure

Sketch preprocessing converts an inaccurate sketch into an edge-vertex graph, which is a rough projection of a 3D scene. Sketch strokes are smoothed, and then the entities are linked at their endpoints to form a connected graph.

The inputs for the topological analysis are an edge vertex graph together with the 2D coordinates of the vertices. The output is the list of face loops, each loop being an ordered list of vertices.

The 3D model is recovered automatically from the sketch using optimization based methods [Marill 1991; Leclerc 1992; Lipson 1996; Company 2004; Varley

2005]. The appropriate depth value of each vertex is sought by optimizing a compliance function, which is the weighted sum of the constraint functions which account for the relationships among the entities. Constraints are geometric relationships, such as parallelism, perpendicularity and coincidence between two entities, which could be faces, edges or vertices. The constraints are identified from the 2D sketch which holds some implicit spatial information of the 3D object. In the literature, some authors refer to these constraints as regularities.

The 3D model obtained from the reconstruction is usually inaccurate. For example, the faces of the reconstructed 3D model may not be planar; faces that are supposed to be parallel may not be parallel. The model cannot satisfy all the detected constraints partly because some constraints are redundant, inconsistent or even incorrect. For a sketch, the number of such unwanted constraints may be large, which makes it difficult to obtain the intended model that satisfies all the constraints. Consequently, a translation from an inaccurate to an accurate 3D model is necessary before the model can be used for further detailed design. This “beautification”—the process of tidying up the geometry of the model recovered and giving it proper dimensions—is the focus of the work presented in this thesis.

Beautification differs from previous research on reconstruction. The aim of beautification is to obtain an accurate model, and one method involves identifying the constraints in the model and making sure that they are strictly satisfied. The constraints can be solved by an optimization procedure, such as iteratively moving the vertices in small steps. The input to the beautification process is the 3D model

obtained from the reconstruction. The optimization approach has been used to solve the constraints by scholars such as Varley [Varley 2002a], who beautified the 3D model reconstructed from a line drawing by determining the face normals and distances between faces separately. A similar problem of inaccuracy arises in reverse engineering where a 3D model is recovered from a point cloud [Langbein 2004]. The problems of beautification and reconstruction are not always distinct. Beautification is treated as a geometric constraint satisfaction problem by Varley [Varley 2002a], who subdivided the beautification problem into determination of face normals and determination of face distances from the origin. Some constraints are selected and translated into compliance functions which can be expressed as equations. But there is a theoretical doubt about the subdivision: There are some objects with resolvable representations (the objects can be obtained by solving their corresponding constraint systems), but for which determining face normals first, and then face distances, do not achieve a resolvable representation. It is not clear if it is easy or possible to express some types of constraint by face normal and distance, such as the distance between two vertices.

Dimensioning requires a constraint set with no redundancy. The constraints include topological constraints and constraints identified from the 2D sketch. Martínez and Félez presented a constraint-based solver to provide a completely dimensioned 2D part [Martínez 2005]. But dimensioning a 3D model needs further investigation.

Constraints are used in both the reconstruction and beautification processes

[Lipson 1996; Langbein 2004; Piquer 2004]. One of the constraints that is difficult to establish is skewed symmetry [Lipson 1996; Piquer 2003], which includes mirror and rotational skewed symmetry. The work in this thesis tackles this problem.

1.2 Thesis objectives and scope

The work in this PhD research is part of a project that aims ultimately to create a system that will take a designer's 2D sketch and convert it into the intended 3D object suitable for use within a CAD system. The inaccuracy inherent in a sketch means that the 3D model recovered can only be rough and not quite ready.

The work in this thesis investigates the processes involved in “beautifying” the object, and not the object recovery directly. Stated succinctly, the objective of this research is: given a sketch and the rough 3D object recovered from it, return the actual object intended by the designer.

Beautification involves detecting and selecting the relationships (or constraints) among the entities (faces, edges and vertices) in an object. Many such relationships exist, including parallel faces and edges, orthogonal faces and edges, symmetry etc. Parallelism and orthogonality are relatively easy to detect; symmetry, however, is not. So a fair amount of effort has been spent on symmetry detection, which can be done on both planar and 3D objects [Matsuda 1993; Zabrodsky 1993; Yip 1994; Yip 1999; Yip 2000]. The focus in this thesis is on the detection of symmetry in a 2D sketch, a much more challenging technical problem than from a 3D object.

The other major effort in this thesis is given to detecting and selecting the

constraints in 3D and then solving them, thereby improving the geometry of the rough 3D object and producing one that satisfies these constraints.

1.3 Organization of the thesis

Chapter 2 reviews the existing work in reconstructing 3D objects from 2D sketches, and introduces some of the fundamental ideas therein, such as the use of regularities and constraints. It also sets up the problems which we are to attack in this thesis.

Chapter 3 presents a novel method for detecting skewed mirror symmetry from a 2D sketch. Chapter 4 describes a method for detecting skewed rotational symmetry.

The beautification and dimensioning of the 3D recovered model from a 2D sketch is performed by selecting a proper set of constraints describing the model, which may be detected automatically or be given by the user directly. Chapter 5 proposes a graph-based method to detect the structurally redundant and conflicting constraints. Chapter 6 describes a numerical method for detecting and then removing redundant and conflicting constraints, which results in a sufficient and consistent constraint set required to produce an accurate model. A priority of constraints is introduced to select the detected constraints.

The methods described in Chapter 3 to Chapter 6 have been implemented in a computer program SRLD, “3D Surface Reconstruction from 2D Line Drawing”. Experimental results are presented in the chapters too. Chapter 7 provides a summary of the work done and contributions, followed by a list of future work.

CHAPTER 2

Literature Review and Problem Statement

In traditional CAD systems, geometric modeling is performed by specification of the exact geometry. But in conceptual design, designers often do not have the exact geometry. They often express their ideas through sketches, which are rough and inaccurate, and devoid of dimensions.

Herbert notes that “A sketch provides an extended memory of the visual images in the designer’s mind” [Herbert 1987]. An industrial designer, in relating his experience with existing CAD systems, said: “the interface is just not for us, I can do thirty sketches on the paper by the time it takes me to do two on the computer” [Lipson 1998].

2.1 Literature review on sketching systems

2.1.1 Interactive construction approach

Interactive construction of 3D objects from 2D sketches has been investigated by Fukui [Fukui 1988], Igarashi [Igarashi 2001], Pugh [Pugh 1992], Zeleznik [Zeleznik 1996] and Bloomenthal [Bloomenthal 1998]. Their systems are interactive and practical, but need additional interactive guidance from the user. Fukui [Fukui 1988] developed a system that constructed a 3D object face by face. He constructed a face by referring to the geometry of existing connected faces. The first face is placed in an arbitrary position. Since the whole system is interactive, the user can find and correct the mistakes easily. Igarashi [Igarashi 2001] introduced a suggestive interface with a

list of thumbnails in the bottom of the design window in his 3D drawing system, Chateau. After the user has added a line in the partially finished object and given the system hints about the desired operation by highlighting related components in the scene, the system suggests subsequent operations in an array of small thumbnails showing the possible constructed solids. A set of suggestion engines is used to generate the list of thumbnails. But for a complex object, it is hard to give the right suggestion, and the user has to spend more time on working through the thumbnails because the system requires the user's confirmation after showing multiple suggestions. Pugh [Pugh 1992] described a 3D interactive system, Viking, that uses interactive sketch interpretation to create a "what you draw is what you get" user-interface. Sketches are three-dimensional entities in Pugh's system. The user can position a vertex in two different views by showing where it "should be". The 3D object is consistent with both the designer's line-drawing and a set of geometric constraints either derived from the line-drawing or placed by the designer. By using the techniques of other systems, such as Snap-Dragging [Bier 1990], the constraints can be defined more implicitly based on the user's dragging operation. The system can produce solid models directly, but not in a way natural for the users [Qin 2000].

3D primitives such as blocks and prisms have been used in interactive sketching systems. The SKETCH system [Zelevnik 1996] introduced a purely gesture interface based on simplified line drawings of primitives that are composed of strokes and interactors. 3D geometric primitives are created and placed from their corresponding gestures. Editing operations such as resizing are accomplished by recognizing edit

gestures. Bloomenthal [Bloomenthal 1998] extended Zeleznik's work to design machined metal and plastic prismatic parts in their system, Sketch-N-Make. The system transforms the geometry into a high level feature-based model that represents the geometry and additional information from which machining features are derived. Quick-sketch, described by Eggli [Eggli 1997], partly adapted conventional 3D techniques for a sketch-based environment. The system models a 3D object by rotationally or translationally sweeping a 2D profile about an axis. The 2D profile may be lines or curves sketched on planar faces of existing objects. Ruled surfaces can be defined between two sketched curves also in Quick-sketch. The sketched 3D objects may be refined by defining 2D and 3D geometric constraints.

In the system of Qin [Qin 2000], users can mix free hand sketching, interactive 2D input and 3D primitives input to quickly construct 2D and 3D primitives. A rule-based 3D primitive recognition is conducted by the use of the inference knowledge in terms of matching its 2D primitive configurations and connection relationships to some patterns. The system can recognize the following features: boxes, cylinders, objects of revolution, spheres and other ruled surface features such as swept surfaces, modified features and complex extrusion features.

2.1.2 Line labelling

The interactive approaches introduced above reconstruct the scene step by step. The systems described below interpret an entire line drawing at once.

The edges of a line drawing can be labelled to reflect a consistent 3D interpretation. A significant body of work in line labelling was produced by Clowes

[Clowes 1971], Huffman [Huffman 1971], Waltz [Waltz 1972], and Kanade [Kanade 1981]. The labels could describe the edges as being convex, concave, or occluding. Successful line labelling provides useful information about an object. Then the depth ordering of visible vertices and the hidden topology of the object can be obtained from the junction labels [Varley 2002b]. Although line labelling is useful and well-established, there is no definite way to produce geometrically realizable line labelling, because a valid line labelling of a drawing does not present a valid object, and at the same time a drawing may have many possible line labelling schemes.

Wang and Grinstein [Wang 1989] produced a constructive solid geometry (CSG) representation of the object depicted in a drawing. With the Huffman-Clowes labelling on the drawing, their system can identify the primitive blocks and the set operations necessary for the CSG tree generation. They were interested in the “Y” and “W” type of corners. The line-labels of the junction determined whether the block was to be added or subtracted and the orientation of the junction lines determined the orientation of the block in space by transforming the primitive cube to the blocks. Wang [Wang 1991] extended this method to drawings depicting prismatic objects and tetrahedral primitives. However, Wang’s method is limited in the scope of drawings and objects it can handle.

2.1.3 Linear system approach

Sugihara [Sugihara 1984a, b, 1986], Grimstead [Grimstead 1996] and Varley [Varley 2002b] succeeded in representing an explicit polyhedral 3D model from a line drawing in terms of linear algebra, which is a necessary and sufficient condition for

reconstruction. After using a variant of the Huffman-Clowes labelling scheme to label the drawing, Sugihara [Sugihara 1986] reduced the problem of judging the correctness of a line drawing to a problem of checking the existence of a solution to a certain system of linear equations and linear inequalities. The equations come from the constraint of vertices lying on planar faces. The inequalities come from the relative depth constraints deduced from the line labelling. By combining the visual information on the surfaces of objects, such as shading and texture, with the algebraic structure of line drawings, Sugihara can determine the object shapes uniquely from single-view images.

Grimstead and Martin [Grimstead 1996] reconstructed objects using a system of linear equations where the unknowns are the parameters of the faces and depth value of the vertices. Equations ensuring vertices on the corresponding faces, parallelism of 2D lines and skewed symmetry are included. These equations are solved simultaneously using a least-squares fit algorithm. Varley [Varley 2002b] described a linear system called RIBALD, where the only unknowns are the z-coordinates of the vertices. Varley introduced a new compliance function, junction label pair (JLP), which requires the junction and line labels. From the junction type Varley obtained the ratio of depth change to 2D line length and the constraint equations from the junction type. A black-box linear system solver, Ortholin2, was invoked to find the least-squares fit to the over-constrained system of equations, in which reconstruction is achieved using a single primary compliance function (JLP or corner orthogonality) derived from orthogonality at vertices (that is, the faces intersecting at the vertices are

perpendicular to each other) and supported by secondary compliance functions (line parallelism and/or four-vertex planarity). However, like Sugihara's system, this system is sensitive to inaccuracies, and not all the constraints can be formulated as linear relationships.

2.1.4 Nonlinear system approach

The optimization approach requires a complete wire-frame sketch as input, where all hidden lines are visible [Marill 1991; Leclerc 1992; Lipson 1996]. The optimization methods gradually assign the depth of each vertex from an initially flat drawing of a 3D wire-frame. The fundamental principle of Marill [Marill 1991] underlying the operation of the construction problem is extremely simple: it is to minimize the standard deviation of the angles created between lines at junctions in the constructed object. The key to obtaining a rough model by simply minimizing the angle diversity comes from Attneave and Frost's idea of "minimization of diversity in angles, lengths and slopes" [Marill 1991]. Marill achieved this goal by optimizing the unknown depth coordinates of the drawing vertices while the objective function is the minimum standard deviation of angles (MSDA) at junctions. Marill used a simple steepest descent algorithm of Hill-Climbing to optimize the target function. Leclerc and Fischler [Leclerc 1992] explained why Marill's algorithm was able to perform as well as it did on the examples that he presented: Marill's examples have symmetric planar faces or all angles equal. Leclerc and Fischler improved Marill's algorithm by introducing a face-flatness criteria into the objective function and improved the numerical behavior by implementing the conjugate gradient algorithm to optimize the

target function. Lipson and Shpitalni [Lipson 1996] provided a list of regularities: face planarity, line parallelism, line verticality, isometry, corner orthogonality, skewed facial orthogonality, skewed facial symmetry, line orthogonality, minimum standard deviation of angles, face perpendicularity, prismatic face, line collinearity and planarity of skewed chains. The regularities were weighted according to their degrees of accuracy in freehand drawings. The depth values were obtained by minimizing the compliance function which is a linear sum of the weighted combination of the different regularities. They examined several optimization methods and preferred cyclic application of Brent's method [Brent 1973].

2.1.5 A two-step approach

In order to overcome the problems of local optima associated with earlier approaches to optimization-based 3D reconstruction, Company [Company 2004] and Varley [Varley 2005] proposed an approach consisting of two consecutive steps. After classifying the models based on some simple perceptual rules into normalons, prisms and pyramids, different inflation methods are employed to obtain tentative models in the first stage. Axonometric inflation gives good results for normalon and quasi-normalon models, and level-inflation which is based on vertex typologies is used to obtain tentative models from prismatic models. The tentative models are then improved by optimizing a objective function expressing some other regularities [Company 2004; Varley 2005]. A recent article by Varley [Varley 2005] described a new method which, in the absence of line labels, inflates a drawing to produce a frontal geometry. The initial frontal geometry is created by setting all z-coordinates to

lie on the surface of a cone. In the next step the frontal geometry is iteratively updated using predictions from the compliance functions, including compliance function of face-vertex coplanarity, corner orthogonality, major axis alignment, parallelograms, line parallelism and through lines.

2.2 Constraints

Humans are able to understand sketches depicting three-dimensional objects rapidly and easily. When humans interpret sketches, they make some assumptions regarding the objects depicted based on their prior experience. Constraints are involved in expressing this prior experience. In the related literature, constraints are also known as regularities; they define the geometrical relationships between entities or within a group of entities. The constraints appearing in a 2D sketch correspond to the real geometrical constraints existing in the 3D object depicted.

Some constraints have been used to reconstruct 3D models from 2D sketches. The reconstruction is a process of adding depth to the sketch. Traditional methods of reconstruction use a set of constraints to create a system of equations or compliance functions, where the variables are the depth values of the vertices in the sketch. The best solution to this system corresponds to the 3D model.

The constraints appearing in 2D sketches may be accidental. For example, two parallel lines in a 2D sketch may not represent parallel lines in the 3D object, though it is more likely that they do.

Some constraints may be false or unnecessary. They might be redundant or

inconsistent with other constraints, so a system may be over-constrained. Exploiting the relationships among constraints can reduce the number of constraints. For example, if two faces are both parallel to a third face, then the two faces are parallel to each other. No more constraints need be introduced between them. The constraints present in a sketch are not exact, due to the imprecise nature of sketching. The inaccuracy should be taken into account in detecting, selecting and solving constraints in construction and beautification.

Below is a list of constraints. Each constraint may exist in a 2D sketch and give a hint about the relationship among individual entities in 3D. The number of constraints is large, and using all of them in the 3D model recovering process would result in a large and complex compliance function which is difficult and costly to solve. Furthermore, some of these constraints duplicate each other. Hence, normally only a subset is used. The process of selecting the best subset of constraints has been studied by Sun [Sun 2006].

Face planarity

A designated planar face in a sketch is required to be actually planar in the 3D model [Leclerc 1992; Lipson 1996; Varley 2002b; Company 2004; Varley 2005].

Parallel faces

A parallelism constraint requires two faces to be parallel. Parallelism constraints affect only face normals, not distance between the faces.

Perpendicular faces

A perpendicularity constraint requires two faces to be perpendicular. It can be

deduced from orthogonal corners, i.e. corners with mutually orthogonal faces. For each vertex at an orthogonal corner, three two-way face perpendicularity constraints are generated [Lipson 1996].

Corner orthogonality

Orthogonal corners are common in engineering objects. Lipson found orthogonal corners based on the fact that the projection of an orthogonal corner spans at least 90° , which can be expressed by $\min(\arccos(i_1, i_2), \arccos(i_1, i_3), \arccos(i_2, i_3)) > 90^\circ$, where i_1 , i_2 and i_3 are vectors representing the three lines of the junction and pointing from the junction outwards. A junction of three lines has eight variants, created by flipping the direction of each line and considering the eight resulting permutations. If a three-line junction is a projection of an orthogonal corner, all of its eight variants must span at least 90° [Lipson 1996]. Using line labeling, W-junctions and Y-junctions are interpreted as orthogonal corners by Varley [Varley 2002b; Varley 2005].

Face distance constraint

A face distance constraint fixes the distance between two faces, which also means that the faces must be parallel.

Parallel lines

Lipson [Lipson 1996] observed that a parallel pair of lines in the sketch plane reflects parallelism in 3D.

Vertical lines

A line that is vertical in the sketch plane (parallel to the y axis of the drawing page) is vertical in 3D [Lipson 1996].

Line orthogonality

All line pairs at a junction except those that are collinear are perpendicular in 3D [Lipson 1996]. This regularity is termed MSDP (minimum sum of dot products). Two lines parallel to two different axial directions are considered to be perpendicular [Company 2004].

Line collinearity

Lines collinear in the sketch are collinear in the 3D model [Lipson 1996; Company 2004; Varley 2005].

Parallelograms

A parallelogram in a sketch should correspond to a rectangle in 3D [Varley 2005].

MSDA

Marill [Marill 1991; Leclerc 1992; Lipson 1996] noted that the natural interpretations of convex polyhedra tend to be those with the minimum standard deviation of angles (MSDA) at corners.

Isometry

Lipson suggested that the lengths of entities in a 3D object are uniformly proportional to their lengths in the sketch plane [Lipson 1996], which is the case for isometric drawings.

Skewed symmetry

A skewed symmetry depicts a real symmetry viewed from some direction [Kanade 1981]. Figure 2.1 shows some example of skewed symmetry faces, which are

symmetric in 3D. A face showing skewed symmetry in 2D denotes a truly symmetrical face in 3D [Lipson 1996; Piquer 2004]. Lipson used a simplification method to identify the symmetry that exists in a polygonal shape. Axes of candidate mirror planes in an object can be chained to form a mirror plane of an object [Piquer 2003; Piquer 2004].

Figure 2.1: Faces showing skewed symmetry.

Skewed face orthogonality

If adjacent edges on the contour of a planar face join only at right angles, then the contour is orthogonal. If the face is viewed from an arbitrary viewpoint, then the orthogonality will appear skewed, as shown in Figure 2.2, where the edges of the contours are all parallel to the two main axes. A face contour that shows skewed orthogonality is probably orthogonal in 3D [Lipson 1996; Varley 2002b]. Skewed facial orthogonality is taken as a particular case of line orthogonality extended to all edges of the same face [Company 2004].

Figure 2.2: Faces showing skewed orthogonality.

Prismatic face

A face with two opposite edges joined by parallel straight lines is a prismatic face. For example, two elliptic arcs in a projection plane is a projection of a prismatic face in space, which is cylindrical. As shown in Figure 2.3, the bold lines are two elliptic edges, the face joining the two edges is a prismatic face [Lipson 1996].

Figure 2.3: Two examples of prismatic face.

Planarity of skewed chains

It is a combination of planarity and rotational symmetry or orthogonality when a chain of entities is found to possess skewed symmetry or skewed orthogonality

[Lipson 1996].

Major axis alignment

The three most common line orientations in a drawing correspond to the three major axes of the object [Kang 2004; Masry 2005; Varley 2005]. Lipson [Lipson 1996] detects the three major axes by means of an angle distribution graph.

Junction label pairs

The ratio of change of the depth for one line to its 2D line length is defined according to junction types of the line labeling [Varley 2002b].

Features

A feature is a specific collection of geometric entities within an object. Identification of features in two-dimensional drawings is an item of current research. Meeran and Taib [Meeran 1999] recognized features from 2D orthographic projections through a two-stage process of profile searching and feature completion.

MSDA, isometry, skewed facial orthogonality and planarity of skewed chains are properties of the object as a whole, not local properties. In the beautification process, all the possible constraints are first detected. Then, conflicting and redundant constraints are identified and removed, leaving behind a well-constrained subset, which is then solved to obtain the beautified model.

2.3 Constraint solving

In the beautification process of the system described in this thesis, the 3D model reconstructed from a 2D sketch is required to satisfy the constraints identified. The

task is similar to the constraint solving in parametric CAD design.

A geometric constraint problem consists of a finite set of geometric objects such as vertices, edges and faces, and a finite set of constraints, such as distance and parallelism. Geometric constraint solving finds geometric configurations that satisfy the constraints.

Constraint solving has been proved to be an indispensable tool in many applications such as mechanical part design, kinematics, robotics, molecular modeling and computer vision [Hoffmann 2001a]. In parametric design, geometric constraint solving allows a user to make modifications to existing designs by changing parameter values, where the user just needs to prepare a rough sketch annotated with specific geometric constraints that defined a precise geometric object.

This section gives a short review on the methods of constraint solving. A more substantial review in relation to beautification is presented in Chapter 6, which deals with the topic.

2.3.1 Rule-based approach

A sequence of construction steps for constraint solving, which replaces the whole constraint solving process by a set of sequential sub-steps, similar to building a shape by means of a compass and protractor [Lipson 1998], can be derived using a rule-based or a graph-based technique. The geometric entities in the sequence of construction steps are determined using a symbolic or numerical procedure.

In the rule-based approach, sequences of construction steps are derived by rewriting rules of the constraints expressed by predicates. The rewriting rules are

$\phi \rightarrow \Psi$ where ϕ is a list of known predicates or facts and Ψ is a list of derived predication [Anderl 1996]. If ϕ matches some predicates of the constraint system, then some of the predicates are replaced by new and simpler ones [Bruderlin 1987]. The rewrite processes repeat until for every point in the sketch, a predicate of the position of this point is given or can be derived from other given predicates.

A set of rules for the expert systems is described to solve the problems for 2D designs [Sunde 1987; Aldefeld 1988; Verroust 1992; Gao 1998b]. Lee [Lee 1996] presents a new method for geometry reasoning using graph representation to improve the rule matching process.

Suzuki [Suzuki 1990] developed a rule-based geometric reasoning system using constraints propagation at the geometric element level incorporating ATMS (Assumption based Truth Maintenance System) database techniques. After changing some constraints, a designer can change the shape consistently, which is done by changing assumptions in the ATMS database.

All the rule-based methods have their intrinsic limitations: not all the 2D constrained models can be solved by a finite set of rules; inefficient computation, exhaustive searching and rule matching are the main disadvantages of this approach. [Lee 1996].

2.3.2 Graph-based approach

A geometric constraint system can be represented by a constraint graph which includes nodes and arcs, where a node represents an entity (vertex, edge or face) and an arc between two nodes represents a constraint between the entities of the nodes. By

repeatedly splitting a graph into bi-connected components until the graph cannot be split further, Owen [Owen 1991] derived a sequence of construction steps when the geometry can be constructed using a ruler and a pair of compasses in 2D. Fudos [Bouma 1995; Fudos 1996, 1997; Fudos 1998] hierarchically reduced the system into triangles by clustering the geometric elements. By incrementally identifying a set of constrained geometric entities with three degrees of freedom (DOF) as rigid body, Lee [Lee 1998] solved the geometric constraint problem that is not ruler-and-compass constructible. The graph is reduced to a single node after a sequence of DOF-based graph reductions. In 3D cases, a cluster of elements that depend on each other can be replaced by one node in the graph, and the DOF of the new node is equal to 6 and elements of the cluster will be solved together [Li 2002]. Maximal matching algorithm is used to detect the sub-graphs with $\text{DOF} = 6$ by Li.

A different approach of constraint graph analysis used rigidity theorems. Hoffmann [Hoffmann 1997b; Hoffmann 1998] presented a general max-flow based approach to isolate a subgraph in both 2D and 3D that can be solved separately; such a subgraph is called dense subgraph. Finding a minimum dense subgraph is NP-hard. The constraint system is then solved by iteratively reducing the dense clusters to a single geometric object. Based on the method of Hoffmann [Hoffmann 1997b; Hoffmann 1998], a new decomposition-recombination (DR) planner, MFA (Modified Frontier Algorithm), was proposed by Hoffmann and Sitharam [Hoffmann 2001b; Hoffmann 2004; Sitharam 2004; Sitharam 2006; Oung 2001]. MFA keeps the structure of the frontier vertices of the dense cluster, i.e. those vertices that are

connected to vertices outside the cluster.

In the spatial case, the constraint problem is decomposed into smaller ones according to some basic configurations: tetrahedron and octahedron among points and planes are considered by Durand and Hoffmann [Hoffmann 1995a; Hoffmann 1995b; Durand 2000]; several octahedral problems involving points and lines are also studied by Durand and Hoffmann [Durand 2000; Hoffmann 2000], where symbolic reduction and homotopy continuation methods are used to simplify the equations of constraints. A locus intersection method, a hybrid method based on geometric computation and numerical search, is proposed by Gao [Gao 2004] to solve all the basic configurations.

A graph is decomposed and simplified by removing terminal nodes from the graph [Hsu 1996; Eggli 1997; Chung 2000]. When the total number of degrees of freedom of the edge is smaller or equal to that of the node, a node and its connecting edges can be removed from the graph.

Kramer [Kramer 1992] developed an incremental geometric constraint solver based on degree of freedom analysis, which successfully solves geometric constraint satisfaction problems in the kinematics domain. Dohmen, Noort and Bidarra. [Noort 1997; Dohmen 1998; Noort 1998; Bidarra 1999; Bidarra 2000] implemented a system called SPIFF that can automatically adjust a feature model of a product while ensuring a valid model or keeping the feature model consistent with those of other product views. Their geometric constraint solver is based on the constraint solving approach described by Kramer [Kramer 1992] where each face of the model is mapped onto a geom, which has a position and orientation freedom. Although Kramer's method

employs geometric knowledge in the solving process, its implementation is very elaborate, because there are many plan fragments required [Dohmen 1998]. Using symbolic geometric reasoning based on the work of Kramer in the constraint solving system, Csabai [Csabai 1996] introduced a top down approach for creating the product layout and subsequent detailed design.

In order to beautify the reverse engineered geometric models, Langbein [Langbein 2004] selected consistent constraints by consecutively adding constraints to the graph that begins from one node. Then the constraints are solved using a numerical method to obtain a 3D model representing the ideal design intent. His approach is based on Kramer's degree of freedom analysis and Li's method of analyzing dependencies between geometric objects [Li 2002].

By analyzing the connectivity of entities and constraints [Latham 1996], the maximum b-matching algorithm is used to subdivide a large set of constraints into small subsets that can be solved independently. Similarly, Ait-Aoudia [Ait-Aoudia 1993; Hoffmann 1997a] proposed a method based on maximum matching of the bipartite graph to decompose a well-contained system into irreducible ones which can speed up the resolution in case of reducible systems. Based on Latham's method and Joan-Arinyo's [Joan-Arinyo 2004] idea of s-tree decomposition, Gao [Gao 2006] proposed a C-tree decomposition method to solve a 3D well-constrained problem by reducing it to smaller rigid bodies.

2.3.3 Symbolic approach

In the symbolic approach, constraints are converted into a system of algebraic

equations and then solved using elimination methods such as Grobner basis [Buchberger 1985] or Wu-Ritt's method [Wu 1994]. A Grobner basis of polynomial ideals is used for the solution of systems of algebraic equations. Via the computation of the S-polynomials and polynomial reductions, the equation system is transformed into a triangular one, similar to that produced during Gaussian elimination in the linear case. Wu-Ritt's method presents the zero polynomial sets as the union of the zero polynomial sets in triangular form. Gao [Gao 1998a] gave a complete method to solve constraint systems based on Wu-Ritt's decomposition algorithm.

The Grobner basis method is employed in the constraint-based design [Buchanan 1998], where parts are positioned relatively in terms of the relationships or constraints that implicitly exist between their constituents. In Kondo's system [Kondo 1992], geometric constraints are automatically generated by the interpretation of the modeling operations. With the interactive manipulations such as the canceling, adding or changing of dimensions, the relationships between these constraints are updated using the Grobner basis method.

The symbolic methods keep the possibility of producing several numerical solutions. But the computational cost of the derivation of symbolic approach is heavy in general. The existing work is confined to 2D cases and the extension to 3D cases is still under investigation because of the complexity of this method.

2.3.4 Numerical approach

In this approach, the constraints are translated into a system of equations and then the equations are solved simultaneously using numerical techniques.

Light [Light 1982] presented a method using Duff's sparse matrix method to reduce the time required to solve the equations by decoupling the constraints. The number of constraint equations to be solved is significantly reduced by partitioning the Jacobian matrix used in the Newton-Raphson method. Ge [Ge 1999] employed a BFGS method, which is a quasi-Newton method, to solve the geometric constraints.

In reverse engineering, B-rep models are created from point clouds. Initial models can be generated by constraints solving methods, such as the Levenberg-Marquardt algorithm [Werghi 1999] or Newton-Raphson method [Benko 2002]. The generated initial models are inaccurate and can be improved in the post processing step by selecting and solving a consistent set of constraints using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [Langbein 2002].

The numerical approach is powerful and general, and can solve complex problems that are unsolvable by other methods. But it requires good initial values and most numerical methods have trouble in handling redundant constraints.

2.4 Problem statement

Much research has been dedicated to different mechanisms for constraint solving. Constraint solving in 3D cases is still a research topic under investigation. Due to the complexity of the 3D constraint problem, it is not possible to use the rule-based approach or symbolic approach for reconstruction or beautification. Currently, there is a lack of effective spatial variational constraint solvers for under-constrained systems. Also, determining if a constraint graph is well-constrained, over-constrained or

under-constrained is still an open problem [Sitharam 2004]. Beautification of reconstructed 3D models requires the models to be well-constrained. The establishment of a well-constrained system requires a method for detecting redundant or inconsistent constraints. This research investigates a graph-based method for detecting redundant or inconsistent constraints because of its potential efficiency in execution. But graph-based methods have limitations, a topic which will be discussed in detail in Chapter 6. Therefore a numerical method is also investigated for detecting a well-constrained system in the beautification process.

Currently existing systems of the types described in Section 2.1 can create approximate 3D models [Sugihara 1986; Marill 1991; Leclerc 1992; Lipson 1996; Varley 2002a; Company 2004]. However, the objects obtained from the reconstruction seldom strictly conform to all the constraints. In practice, if a 3D object is reconstructed from a 2D sketch, the coordinates of the vertices will seldom lie in the right places because a sketch is inherently inaccurate. Therefore a rectification process, known as beautification in the literature, is necessary after 3D reconstruction because the model must finally be made accurate if it is to be used later for detailed design. We aim to beautify the model by finding and solving the constraints in the object automatically.

Though constraints are used in reconstruction too, it can yield the correct result only when all the parts in the optimization objective function are zero, both in Sugihara's linear system approach and Lipson's nonlinear approach. The method of reconstructing three-dimensional solid models has not been very successful because

the number of constraint equations required to fully constrain all the topological entities in a complex 3D model can be very large and pose problems, including redundant and inconsistent constraints, to the numerical procedure.

Geometric constraints are detected firstly from a sketch or a reconstructed model and then a consistent set is imposed from the detected constraints to obtain an improved model. A large number of constraints are likely to over-constrain the model or introduce inconsistencies. An appropriate and consistent subset needs to be selected from the detected constraints to form a solvable constraint system. Then a constraint solver is required to compute a solution to the constraint system so that an improved model can be obtained.

The models created by the reconstruction process are initial models, and serve as the input for the beautification. In the process of beautification, we can tell whether a small shift in the structure, for example a slight reorientation of a plane, would cause the constraints to be met. A new object can then be constructed that satisfies these constraints. This new model is expected to be a better representation of the object.

Symmetry is a constraint that can help in both the reconstruction and beautification processes. The detection of skewed symmetry, both mirror and rotational, from a drawing will be tackled first. This is the subject of the next two chapters.

The main focus of this thesis is on beautification. In order to do beautification, certain sub-issues need to be dealt with; symmetry detection and constraints selection are two of the main ones covered in this thesis. Symmetry, covered in Chapters 3 & 4,

is a constraint that contributes to the beautification process. The constraint selection analysis covered in Chapter 5 & 6 is used to select and solve the constraints for beatification.

CHAPTER 3

Skewed Mirror Symmetry Detection from a 2D Sketch of a 3D Model

Mirror symmetry is an important constraint in the 3D reconstruction of an object from a 2D sketch and the subsequent beautification of the 3D model. The “mirror” in the symmetry is a plane, called the symmetry plane, about which the object is symmetric (see Figure 3.1(b)). This chapter proposes a new method to detect symmetry planes from a sketch by exploiting the topological connections of the edges it contains. Experiments show that the method can detect all the symmetry planes and the corresponding symmetric vertex pairs, edge pairs and face pairs as well.

3.1 Introduction

Many constraints need to be detected for the 2D to 3D reconstruction [Lipson 1996]. Information on symmetry can lead to a significant speedup of the constraint finding process, avoid redundant constraints and improve the efficiency of the 3D model recovery. Symmetry analysis has many other applications, especially in artificial intelligence and related fields. Symmetry helps humans to interpret drawings of physical objects. A large number of industrial components have some symmetry also.

Figure 3.1 illustrates the three forms of geometric symmetry: 2D, 3D and skewed. 2D geometric symmetry is line reflection in 2D; a line reflection occurs when a set is

reflected about a line to produce the symmetric set. 3D geometric symmetry is reflection about a plane in 3D. 2D and 3D geometric symmetries are also called bilateral symmetry or mirror symmetry. Plane reflection occurs when the entire set is reflected about a plane to produce the symmetric set. Most research has been concerned with mirror and rotational symmetry. The most common symmetry in our environment is 3D mirror symmetry.

Figure 3.1: Three forms of geometric symmetry.

Skewed symmetry, as defined by Kanade [Kanade 1981], depicts a real symmetry viewed from some (unknown) viewing direction. A face showing skewed symmetry in 2D denotes a truly symmetrical face in 3D. A symmetric face is transformed during 2D projection. Figure 3.1(c) shows some skewed symmetric faces after projection. A skewed spatial symmetry denotes a truly symmetrical model in 3D (Figure 3.1d).

In this chapter, we present a novel method for detecting the skewed symmetry planes of an object from its 2D line drawing. The next section introduces the problem definition and assumptions, followed by a review of the related work. The algorithm for detecting symmetry is then developed. Finally examples from our implementation

are given.

3.2 Assumptions

The input to our system is a single 2D sketch of some 3D polyhedral object in the form of a 2D edge-vertex graph, including the x, y coordinates of the vertices. The objective is to detect the symmetry plane(s) of the object. No artificial restrictions, such as order of drawing the edges, are imposed on the drawing process. No user input is required other than the sketch itself. Freehand sketches are often rough and inaccurate, which make them fast for conveying formative ideas. Fault tolerance is required to embrace this quality.

Three assumptions are required:

1. All the lines in the 2D drawing are edges. All the edges of the 3D object are drawn.
2. The drawing depicts a view of the object from a general viewpoint that reveals all the edges and vertices, none of which coincide accidentally, and none accidentally appear to be joined in the projection. No edge is projected into a vertex and no face is projected into an edge. Consequently, every edge in the drawing corresponds to exactly one edge in the object.
3. The drawing is a parallel projection of the object.

The first assumption may seem unduly harsh, given that designers do not always sketch all the edges. It is a question of whether we want the algorithm to operate based on “facts”, which means the edges, and thus be deterministic, or to “guess”

what is missing, and thus be non-deterministic. We take the deterministic approach since, to fully describe an object, all the edges will eventually have to be drawn.

There are two types of projections, parallel and perspective, which are used in drawings. We are limiting our drawings to parallel projection because techniques exist to transform perspective projection into parallel. One was developed by Morgan to handle images captured by high altitude imaging systems [Morgan 2004]. Also, “perspective correction” is a popular technique used in commercial imaging systems, such as Photoshop, to transform distorted perspective images to parallel ones [Adobe]. Hence perspective views can be converted into parallel views before applying our algorithm to find the axis of symmetry.

3.3 Previous work

Researchers have shown that the detection of symmetry is important in computer vision. There has been recent work addressing the problem of mirror symmetry detection from range image in computer vision [Parui 1983; Marola 1989; Nalwa 1989; Sun 1999]. Algorithms to detect symmetry cannot be used to detect skewed symmetry directly, since the constraints on 2D symmetry detection are length, gradient vectors and the moments of the contours on both sides of the symmetrical axis, which are not those in skewed symmetrical patterns.

Oh [Oh 1988] and Yip [Yip 2000] used the Hough type voting method to detect 2D skewed symmetry in an image. An advantage of the Hough transform based method is that it can detect skewed symmetries formed by separate curves or points

under noise or occlusion. But at the same time, the method is very time consuming.

An area-based shape representation which is affine invariant is applied to detect rotational skewed symmetries and mirror skewed symmetries [Shen 2001]. The complexity of the algorithm is $O(n^2)$, where n is the number of boundary points for area calculation of the given shape. This method can be used to detect the skewed symmetry from a planar shape only. The method of elliptic Fourier descriptors is applied to detect skewed symmetry under parallel projection [Yip 1994]. A 2D closed contour is expressed by means of a Fourier expansion with length parameterization. Some properties of Fourier coefficients in rotationally and mirror symmetric figures are used to determine symmetry. But because such a parameterization involves calculation of integrals, it is complicated. This method can only detect symmetry in 2D and it is assumed that the geometry is captured accurately in orthographic or perspective projection.

Sugimoto [Sugimoto 1994] provided a method to find a skewed symmetrical pattern as a higher order primitive which is defined by lower order primitives such as line and curved segments in a 2D image. Some rules were used to detect the axis of the skewed symmetrical pattern based on the positions of the segment pairs.

A very simple and effective method for skewed symmetry detection in a polygonal shape was presented by Lipson and Shpitalni [Lipson 1996]. But this method cannot detect the symmetry plane of a model from its 2D projection. The method will be explained and used in Section 3.5.

Zabrodsky [Zabrodsky 1992; Zabrodsky 1993; Zabrodsky 1994; Zabrodsky 1995]

introduced the term “symmetry distance” for measuring mirror and rotational symmetry of objects. The symmetry distance of an object is defined as the minimum distortion required to transform the object into a symmetric one. It is the mean of the square of the distances between the points in the original shape and their corresponding positions in the symmetric shape, and is calculated from the symmetry transform. Although it is theoretically possible to extend this approach to 3D, its complexity makes it impractical for large 3D data sets.

Piquer [Piquer 2003] developed a method for determining planes of mirror symmetry of polyhedral objects based on prior detection of symmetry axes of planar faces. But she eliminated candidate axes of symmetry connecting two vertices across a single face, which will eliminate some real planes of symmetry. It is the most serious limitation of her method.

Figure 3.2: Two isomorphic graphs G1 and G2.

O'Mara [O'Mara 2002] detected hyperplanes of bilateral symmetry using centroids and eigenvectors of the covariance matrix for 2D and 3D objects. He used the method in measuring the symmetry from n-dimensional point data of the surface of a human head. Measuring of global bilateral symmetry about a hyperplane for a set of n-dimensional data points is needed after calculating for the hyperplane. Inability to

find more than n candidate hyperplanes for an n -dimensional data is a weakness of the technique.

Two graphs that contain the same number of vertices and connected in the same way are said to be isomorphic [Gould 1988]. Let $V(G)$ be the vertex set of a simple graph and $E(G)$ its edge set. Then a graph isomorphism from a simple graph G to a simple graph H is a bijection $f: V(G) \rightarrow V(H)$ such that $uv \in E(G)$ if $f(u)f(v) \in E(H)$. For example, in Figure 3.2, isomorphism from G_1 to G_2 is given by $f(a) = a'$, $f(b) = b'$, $f(c) = c'$, $f(d) = d'$ and $f(e) = e'$. An automorphism of a graph is a graph isomorphism of itself. The automorphism group of a graph is also called the symmetric group. The graph isomorphism problem has drawn a great deal of attention over the years, and so far, no completely satisfactory solution has been obtained. Automorphism is a prerequisite of mirror symmetry of a model graph, which must be topologically automorphic if it is mirror symmetric.

One technique often employed in detecting graph isomorphism is known as vertex classification. In this approach, vertices are partitioned according to some graph properties that are invariant under isomorphism. Zabrodsky [Zabrodsky 1993] detected isomorphism by the valency of a point (the number of lines converging at a point) and second order connectivity (the valency of its neighboring points). This method is easy to understand and implement, but unfortunately, valencies alone are not always sufficient. Ronald Gould [Gould 1988; Skiena 1990] used valency in the initial partitioning of a vertex set, and then partitioned the vertices using the relationship of a single vertex to all other vertices. The classification of a vertex v is

based on a distance matrix that represents the number of vertices at each “distance” away from v , a distance being the length of the shortest path, i.e. the smallest number of intervening edges, between two vertices. The distance matrix is used in a backtracking procedure to reduce the search tree of possible mappings. From our experiment, Gould’s initial partitioning does not work for some simple models such as a cube, where each vertex has the same number of vertices at a given distance away from v . The backtracking method is time consuming and does not use the information about the faces of a model.

Nauty [Mckay 1981, 1990] is a publicly available tool for symmetry extraction and graph isomorphism detection using a method based on computational group theory. To achieve such success, Nauty is completely general-purpose, assuming no properties of its input beyond being a colored undirected graph [Darga 2004]. And for almost all model graphs, the number of automorphism solutions is higher than that of the mirror symmetry groups. For example, in Figure 3.9, 19 automorphisms of the model are detected, but just 6 of them are symmetry groups. Although automorphism is a prerequisite for mirror symmetry of the model graph, another method is needed to select the symmetry groups of a model graph from the number of automorphisms.

It is always possible to construct a cyclic directed path passing through each edge once and only once in each direction. Weinberg [Weinberg 1966] added a numbering scheme to the path generation process so that a code is obtained. He proved there exists an isomorphism between two vertices of the graph only if they have the same code. Jiang [Jiang 1992] extended Weinberg’s algorithm for determining isomorphism

of planar triply connected graphs by using a transformation matrix defined by the first three vertices in the path. But his method needs the 3D coordinates of the vertices.

3.4 Topological analysis for face identification

Figure 3.3: A model and circuits corresponding to actual faces

Figure 3.3 illustrates a single graph representing a projection of a 3D wire frame model. A topological analysis can generate a list of circuits of the graph corresponding to the actual faces of the most plausible 3D object depicted by the graph [Leclerc 1992; Shpitalni 1996; Liu 2001; Liu 2002; Liu 2005].

Leclerc and Fischler [Leclerc 1992] identified 3D planar faces in a 2D line drawing of a wire frame object with these rules: (1) Parallel lines in line drawing are parallel in 3D space; (2) A circuit free of internal circuits and internal lines is likely to correspond to a face in 3D. But their method operates only for a limited range of objects.

Figure 3.4: Potential faces of the depicted object

Shpitalni and Lipson [Shpitalni 1996] developed a more efficient method for the general manifold or non-manifold objects. Gibson’s algorithm was used to find a circuit space [Gibson 1985], which contains all the possible circuits embedded in a graph. An example is shown in Figure 3.4. They form the set of potential faces by eliminating implausible faces from the circuit space using rules such as the face adjacency theorem, which states that if two adjacent faces share more than one edge, the edges must lie on the same curve. Finally, they use the A* algorithm [Rich 1991] to search for the optimal set of the potential faces.

Liu and Lee [Liu 2001; Liu 2002; Liu 2005] developed a much faster algorithm to find the faces. They used a depth-first search algorithm to generate the set of minimal potential faces of a drawing, from which a minimum weight clique algorithm is used to select the best face configurations.

The inputs for the topological analysis are an edge vertex graph together with the 2D coordinates of the vertices. The output is the list of face loops, each loop being an ordered list of vertices. In this thesis, we use Liu and Lee’s algorithm [Liu 2001; Liu 2002] to identify the faces in the 2D line Drawing.

3.5 Detecting skewed symmetry of polygonal shapes and rules for checking skewed symmetry

Figure 3.5: Candidate face symmetry axes

As Lipson pointed out [Lipson 1996], if skewed symmetry exists in a polygon, its axis intersects the contour at two points, each either a vertex or the mid-point of an edge. Because the number of vertices and edges on the two sides of the symmetry axis of a symmetrical shape must be equal, the number of possible axes of symmetry is the same as the number of vertices. Let V_i denote the i th vertex of the polygon ($i = 0, 1, 2, \dots, n$), where n is the number of vertices in the polygon and $V_n = V_0$. Each possible candidate symmetry axis passes through the vertices V_k and $V_{k+n/2}$, where $k = 0, 1/2, 2/2, 3/2, \dots, (n-1)/2$. For example, $V_{2+1/2} = (V_2 + V_3)/2$ is the mid-point of the edge joining V_2 and V_3 . So there are five candidate axes for the pentagon shown in Figure 3.5.

3.5.1 Rules for skewed symmetry in a polygon

To check whether a polygon is symmetric, two conditions are required:

1. Two mirror symmetric vertices are equi-distant from the symmetry axis and the line joining them is perpendicular to the symmetry plane.
2. If every mirror symmetric pair of 2D vertices is connected by a line in the 2D

plane, then all these lines must be parallel, i.e., meet the symmetry axis at the same angle.

Let s_d be the symmetric distance of Condition 1. Let $d1_i$ and $d2_i$ be the distances of the two vertices in the i th vertex pair to the axis, then

$$s_d = \text{Maximum} \left| \frac{d1_i - d2_i}{d1_i + d2_i} \right|, \quad (i = 0, \dots, k-1)$$

where k is the number of symmetry vertex pairs in the polygon for the axis being checked. Because a sketch may not be precise, a tolerance must be applied. In our implementation, we allow a 20% error in distance difference, i.e. when s_d is less than 20%, we consider the detected polygon symmetric about the axis.

We define the term s_a to evaluate if angles in Condition 2 are the same. Let θ_i denote the angle of the i th segment, the ends of which are the two vertices of the i th vertex pair, then s_a is the variance of the angles of the symmetry segments.

$$s_a = \frac{\sum_{i=0}^k (\theta_i - \bar{\theta})^2}{k},$$

where $\theta_i = \cos^{-1} \left(\frac{y_{1i} - y_{2i}}{\sqrt{(x_{1i} - x_{2i})^2 + (y_{1i} - y_{2i})^2}} \right)$

and $\bar{\theta} = \frac{\sum_{i=0}^k \theta_i}{k},$

and $(x_{1i}, y_{1i}), (x_{2i}, y_{2i})$ are the coordinates of the i th vertex pair.

When average $(\theta_i - \bar{\theta})$ is less than 15 degree, then $s_a < 0.075$, we consider the polygon symmetric. After the detection of skewed symmetry polygons, we proceed to detect the skewed symmetry plane of the model from the axes detected.

3.5.2 Rules for skewed symmetry in a polyhedron

In a mirror symmetric 3D object, the two 3D vertices in a symmetric vertex pair are equidistant from the symmetry plane and the line connecting the two vertices is perpendicular to the symmetry plane. But in a 2D sketch of a 3D mirror symmetric object, because of the absence of depth value in vertices, only the angle constraint can be satisfied. Equal distances to a plane in 3D project to equal distances in the projection plane only when the projection plane is perpendicular to the symmetry plane. The absence of depth values is the main reason that distance cannot be used as a constraint. So we use the term s_a to check if the 3D object is skewed symmetric, and use s_a to check if two faces are skewed symmetric also. Having the same number of edges in two symmetric faces is an additional rule for checking if two faces are skewed symmetric.

3.6 Detecting symmetry plane in a model

Given a graph corresponding to a 3D model and the faces detected through a topological analysis, it is possible to detect the symmetric vertex pairs, edge pairs and face pairs of the model. If skewed symmetry exists in a model, its symmetry plane intersects the model forming one or more closed sequence of edges lying in the faces of the model. Each end of such edges is either a vertex or the mid-point of an edge. The edge sequence in the symmetry plane will be detected from the 2D sketch of the 3D model as well. This sequence forms the **symmetry polygon**, which is the boundary of the intersection between the symmetry plane and the object.

The skewed symmetry plane of a model must contain edges or the skewed symmetry axes of the faces of the model. We can therefore begin our computation from the skewed symmetry axes of the faces detected using the method described in Section 3.5.

Figure 3.6: Types of edges. (a) End points of axis are mid-point and vertex. (b) End points of axis are two mid-points. (c) End points of axis are two vertices. (d) Axis is an edge.

Each vertex of a symmetry polygon is either a vertex or the mid-point of an edge in the sketch. Figure 3.6 lists all the types of edges in a symmetry polygon according to the type of end points. From Figure 3.6, we can draw a conclusion which we shall call **Property 1**: For every symmetry axis in the edges of the symmetry polygon, at least one symmetric edge pair must exist in the faces adjacent to the face containing the symmetry axis, and the two edges in the edge pair are not the same edge. As shown in Figure 3.6, l_1 and l_1' , l_2 and l_2' , l_3 and l_3' are symmetric edge pairs corresponding to the symmetry axes shown in dashed lines.

In a 3D model, if two planar faces are adjacent, then they only have one line of intersection, i.e. each edge lies in two faces. This is an important property of the graph of a model, and we will be using it in our algorithm. First, we detect all the symmetry axes of all the faces. Each edge of the symmetry polygon is either an axis of a

symmetric face or an edge of the model. So combining the axes detected with all the edges of the model, we define a candidate axes group $Axes = \{axis_1, axis_2, \dots, axis_m\}$.

Take one axis, $axis_i$ of a face from $Axes$. Assume $axis_i$ is contained in a symmetry polygon plane of the model. Then the algorithm below checks if the assumption is correct by analyzing the topological connections of the graph.

Figure 3.7: Edge pairs and face pairs.

We use the variables EP , FP and VP to store the groups of symmetric edge pairs, symmetric face pairs, and symmetric vertex pairs associated with the axis and initialize them to null: $EP = \{\emptyset\}$, $FP = \{\emptyset\}$ and $VP = \{\emptyset\}$.

Beginning from the axis, we identify a set of symmetric edge pairs (see Figure 3.6), if it exists, and add it to EP . From each edge pair, we can find a new face pair which is adjacent to the edges in the edge pair. A face is adjacent to an edge when the edge forms part of the boundary to the face. For example, in Figure 3.7, let the first edge pair detected from the symmetry axis (in dash) be l_1 and l_1' . Edge l_1 is adjacent to Face f_1 and Edge l_1' is adjacent to Face f_1' . So f_1 and f_1' form a potential symmetric face pair if the number of vertices of the two faces are the same, and every edge in f_1

is symmetric to the corresponding edge in f_1' . If the face pair is skewed symmetric satisfying the rules of the last section, then this new face pair is added to FP and all the new edge pairs are added to EP . The vertices of the edge pairs are added to VP as well. After that, another new edge pair will be picked from EP , and faces adjacent to the edge pair are detected again, and the process repeats. In Figure 3.7, let edges l_2 and l_2' be one of the new edge pairs found in the face pair f_1 and f_1' respectively. Then in the next step, from the new edge pair l_2 and l_2' , a new potential face pair f_2 and f_2' is detected. The algorithm ends when all the faces adjacent to the edge pairs in EP are in FP or when a potential face pair is proved to be unsymmetrical. In the former, the model is symmetric. In the latter, it is not.

When a symmetric polygon exists in the model containing the given axis, the algorithm can find all the face pairs. The algorithm ends with a set of symmetric face pairs in the symmetric model. All the faces in the object must be in the set of symmetric face pairs. Existence of a face outside this set means that the object is not symmetric. This does not mean, however, that the model must have an even number of faces, because a single face can be symmetric about the symmetry plane.

If the axis is in the symmetry polygon, our algorithm will not end with null FP , EP and VP . After detecting the symmetric pairs, the edge sequence of the symmetry polygon can be generated easily from the set in *Axes* and displayed to highlight the symmetry. There are two rules for detecting the edge sequence of the symmetry polygon:

1. If the two faces in a face pair are the same, then the face must contain one edge

of the symmetry polygon. The edge is one of the symmetry axes of the face. We can detect the axis of the face easily from the connection of the vertices of the vertex pairs in the face. In the example below, if the algorithm begins from Edge ab in the symmetry polygon $abnm$, then in the next step, Face $abcf$ and Face $abde$ will be detected as a symmetric face pair. After that, Face $cdef$ is detected to be symmetric with itself. So, the Face $cdef$ contains one edge mn in the symmetry polygon.

Figure 3.8: Symmetry polygon.

2. If the two faces in a face pair intersect at an edge, then the edge is an edge of the symmetry polygon.

An axis may or may not be contained in the symmetry polygon. If it is, then all the axes or edges of the polygon will be deleted from *Axes* to avoid redundant computation subsequently. Otherwise, only the axis itself is deleted from *Axes*.

Every axis in *Axes* is checked one by one, to see if it is in a symmetry polygon. The process terminates when all the axes have been checked.

In summary, the steps of our algorithm for skewed symmetry detection in a 2D sketch are:

1. Detect the candidate symmetry axes of all the faces of a model. Check the

symmetry of every face using s_a and s_d . Add the symmetry axes into the group *Axes*. Add all the edges of the model to *Axes* as well.

2. Pop one axis from *Axes*. Set *EP*, *FP* and *VP* to null.
3. Find symmetric edge pairs in the faces of the axis and add them to *EP*.
4. Detect symmetric faces from *EP*.
 - 4.1. If all the pairs in *EP* have been processed, goto Step 5, else pick a new edge pair from *EP*.
 - 4.2. For this edge pair, if the two faces adjacent to the corresponding two edges in edge pair are not in *FP*, we check whether the number of vertices of the two faces are the same and whether the two faces are skewed symmetric using s_d .
 - 4.3. If the two faces in the candidate face pair are not symmetric, then there is no symmetry polygon containing the axis, go to Step 5.
 - 4.4. Otherwise, the face pair is added to *FP*, and the corresponding edge pairs and vertex pairs are added to *EP* and *VP*.
5. If the axis is in a symmetry polygon of the model, the symmetry polygon is saved and all the axes and edges of the symmetry polygon are deleted from *Axes*. Otherwise, delete the axis from *Axes*. Go to Step 2.

Figure 3.9: A model and all its symmetry planes if we run the algorithm without the rules for checking symmetry.

If we do not use the rules for checking symmetry polygon in Step 1 and check for symmetric face pairs in the Step 5 of the summary above, we can detect a set of topologically symmetric planes. Figure 3.9 shows a prism and the result of detecting symmetry planes without using symmetry checking. In fact, the six symmetries detected are the six corresponding automorphism groups of the model graph. Figure 3.10 shows another example; the topology of this model is the same as the one in Figure 3.9. The two examples have the same automorphism groups. After applying the rules of symmetry checking, in Steps 1 and 4 of the summary, we can find the correct symmetry plane quickly, without detecting the automorphism groups. This can significantly speed up the process of detecting symmetry planes. Some axes can be found not to be in a symmetry polygon at an early stage of the algorithm. In the example of Figure 3.10, no symmetry axis can be found for Face f_1 using the rules of Section 3.5. As for axes in other planes, we can find that the candidate face pairs deduced from the axes are not symmetrical just by the number of edges in the faces in the face pairs. By using the rules for checking symmetry, the run time of the algorithm is about one third of the algorithm not using it.

Figure 3.10: Another example with the same adjacency matrix as the example in Figure 3.9

3.7 Complexity of symmetry detecting algorithm

In the worst case, there are $O(m)$ candidate axes of faces, where m is the number of edges in the sketch. From each axis, it needs $O(m)$ time to compute if there exists a skewed symmetry plane containing the axis. Therefore the symmetry detection algorithm runs in $O(m^2)$ time. We may calculate the time complexity in this way instead: At first, Step 2 in the summary of the algorithm runs over all the axes for a time proportional to m . After that, according to the topological connections of the faces and edges only, i.e. without checking if the two faces in a face pair are symmetric, all the faces can be grouped into two groups, in the worst case. All the face pairs are detected. This step runs for a time proportional to m . This is followed by checking if every face pair in the two groups is symmetric, this runs another time proportional to m . So the overall time is $O(m) * (O(m) + O(m))$, is $O(m^2)$. Practical cases are expected to be substantially better than the worst case, as explain above. The results shown in Figure 3.11 demonstrate that the computation time is reasonable

Original 2D Sketch	Sketch with Symmetry Plane Detected	Original 2D Sketch	Sketch with Symmetry Plane Detected
0.003s		0.004s
0.003s		0.002s
0.005s		0.002s
0.004s		0.005s
0.002s		0.005s

Figure 3.11: Some examples of input sketches and their skewed symmetry planes.

3.8 Results

Figure 3.11 shows some results of detecting skewed symmetry planes on a variety of sketches representing objects of various types. As seen in the examples, the algorithm can detect skewed symmetry correctly. The dashed close loops are the symmetry polygons detected. The computation time for the examples is less than 0.005 sec using a Pentium 4 PC.

The method is dependent on the graph of the sketch and the faces detected from the topological analysis process. So incorrect connection of graphs and detection of faces will affect the behavior of our method. In cases of large inaccuracies in a sketch, our method may not detect some of the symmetry planes. This can be overcome by the user identifying one of the edges of the symmetry polygon, which may be the axis of a face or an edge of the model. Then without using the rules of symmetry checking, our algorithm can detect the symmetry plane that contains the input edge.

3.9 Conclusions and discussions

A new algorithm for the detection of the skewed symmetry planes in a sketch of a 3D polyhedral model has been presented. The algorithm has been implemented in our software for the three-dimensional reconstruction of geometric models from sketches. Results of our experiments show that the algorithm works well, with adjustable fault tolerance.

In the algorithm, when checking for symmetry in the 3D model from its 2D sketch, only the angle constraint is required to be satisfied. That is, in the model, if all

the segments which connect two vertices in the vertex pairs are parallel within the tolerance, then the model is considered symmetric. We omit “the equal distance” constraint—the two 3D vertices in a symmetric vertex pair have the same distance to the symmetry plane, because of the absence of depth value of the 3D model in the sketch. The exclusion can lead to a wrong result in symmetry detection for some models. For example, in Figure 3.12, all the line segments (dotted lines in the Figure) connecting the candidate symmetric vertex pairs are parallel. The algorithm will return the symmetry for such models.

The drawing in Figure 3.12, depending on the viewing angle, could be the projection of a symmetric object, and it could also be the projection of a non-symmetric object. So the algorithm returning symmetry is not really wrong, as the actual physical state is ambiguous.

Figure 3.12: An example of ambiguity. The line segments (dotted lines) connecting the corresponding vertex pairs are parallel. The actual object in 3D could be either symmetric or non-symmetric.

CHAPTER 4

Skewed Rotational Symmetry Detection from a 2D Line Drawing of a 3D Polyhedral Object

This chapter introduces a new algorithm for detecting skewed rotational symmetry in a 2D line drawing of a 3D polyhedral object by locating all possible axes of symmetry. As in the case of mirror symmetry, a sketch is first converted into an edge-vertex graph from which the algorithm finds the faces of the object and the sets of topologically symmetric edges and vertices. The algorithm then checks that each set of vertices is rotationally symmetric by analyzing the distribution of the vertices around the circumference of the corresponding best-fit ellipse. The object is rotationally skewed symmetric if the best fit ellipses of all the vertex sets have parallel axes, equal ratio of major radius to minor radius and centres on the axis of rotation. A tolerance is used in the calculation to allow for inaccuracies in the line drawings. A set of experimental results is presented showing that the algorithm works well.

4.1 Introduction

Most research on symmetry has been concerned with mirror and rotational symmetry [Leou 1987; Marola 1989; Cham 1995; Lei 1998; Piquer 2003]. In this chapter, we concentrate on rotational symmetry. Figure 4.1 illustrates the three classes of rotational symmetry: 2D, 3D and skewed.

Figure 4.1: Examples of rotational symmetry.

Figure 4.1c shows some skewed rotational symmetric faces after projection. A skewed spatial symmetry denotes a truly rotationally symmetric model in 3D (Figure 4.1d).

This chapter presents a method for detecting the skewed rotational symmetry of a 3D object from its 2D line drawing. The assumptions are the same as those for mirror symmetry, as stated in Section 3.2. The next section reviews the related work. Subsequently, the algorithm for detecting rotational symmetry is developed. Finally examples from our implementation are given.

4.2 Previous work

A method for detecting rotational and mirror symmetry of planar 2D figures was presented by Matsuda [Matsuda 1993]. Some rigid transformations are applied to the original image; the one that produces a good match between the transformed and the original image is the solution. Leou [Leou 1987] proposed a method to determine the rotational symmetry of a given shape, S , by analyzing the intersection points between S and a circle, C , that has the same centroid as S . However, the methods cannot be

used to detect skewed symmetry.

Skewed rotational symmetry has been studied fairly widely. Often the work is confined to objects of a limited domain, such as planar shapes only [Yip 1994; Yip 1999; Shen 2001], objects of revolution [Glachet 1991] or right straight homogeneous generalized cylinders [Xu 1992]. These algorithms exploit the characteristics of the limited domain, and are not readily applicable to 3D objects outside the domain.

Shen et al. [Shen 2001] applied an area-based shape representation which is affine invariant to detect rotational and mirror skewed symmetries. Yip [Yip 1999] introduced the Hough transform algorithm to detect rotational symmetries from the symmetric patterns that constrain missing data due to noise or occlusion. He proposed a four-pass detection technique to detect centers of rotational symmetry, projection angles, order of the rotation and relevant image points one by one. The algorithm is potentially time consuming. The method used by Yip [Yip 1994] for detecting skewed mirror and rotational symmetry has been introduced in the previous chapter.

No work reported in the literature addresses directly the problem this chapter is tackling, that is, establishing the rotational symmetry of a 3D object given its 2D line drawing and locate the symmetry axis.

4.3 Geometric analysis for skewed rotational symmetry in a vertex set

Our algorithm has two major components, one topological, the other geometric. As will be explained later, if an 3D object is rotationally symmetric about an axis,

then every face of the object must be a member of a collection of faces which is rotationally symmetric about the axis. A collection may have one or more members and an object may have one or more collections. The topological analysis first finds the faces of the object from the line drawing, and then determines if each is a member of such a collection topologically. Having a face not belonging to any collection renders the object non-symmetric. Topologically symmetric means that the number of faces, edges and vertices and the connectivities between them are the same, independent of the geometry. It is computationally cheaper to check topological symmetry first. Only after the establishment of topological symmetry would the more expensive geometrical symmetry be done.

The geometric analysis determines if a set of topologically symmetric faces is geometrically skewed symmetric by checking the physical distribution of the vertices. The geometric analysis is given in this section and the topological analysis in the next.

The geometric analysis uses seven rules to check if a set of N vertices in a drawing is rotationally skewed symmetric. These seven rules are derived from the properties of symmetric vertices in real 3D space. There are five such properties. A set of points lying in one plane in 3D is rotationally symmetric about an axis if the points remain invariant after a rotation through a specific angle, which we call the *symmetry angle*, about that axis. This fact leads to the first two properties:

Property 1: The points must lie on the circumference of a circle.

Property 2: The points must be evenly distributed on the circumference of the circle.

Furthermore, the axis of symmetry is the axis of the circle, normal to the plane

containing the circle and the symmetry angle is $2\pi/n$, where n is the number of points.

Multiple sets of points lying on the same or different planes may together be rotationally symmetric if the point sets are individually rotationally symmetric, and about the same axis. For symmetry, they must possess these three properties (see Figure 4.2):

Property 3: If the different sets of rotationally symmetric points lie on the same plane, then the circles on which they lie must be concentric.

Property 4: Different sets of rotationally symmetric points lying on different planes must have the same axis of symmetry. Hence the centres of their circles must be collinear.

Property 5: The planes of the circles must be perpendicular to the axis of symmetry. Therefore, the circles must lie in parallel planes.

Figure 4.2: A rotationally symmetric object and the circles containing symmetric vertices. Circles 1 and 2 lie in the same plane and are concentric, so are Circles 4 and 5. Their centres are collinear with the centre of Circle 3 on the axis of symmetry, and they all lie in parallel planes.

Hence, the procedure to detect skewed rotational symmetry in a vertex set in a

2D drawing needs to transform the vertices into 3D and check if they are evenly distributed on a circle. The number of vertices in a set can be two or more. For a set with two vertices, it is clear that it must be symmetric, with the two vertices diametrically opposite on a circle, whose center is the mid point between the vertices and whose symmetry angle is π . For a set with more than two vertices, the steps to establish rotational symmetry are as follow:

1. Fit an ellipse to the vertices.
2. If the ellipse exists, obtain the circle whose parallel projection is the ellipse.
3. Map the vertices on the ellipse into vertices on the circle.
4. Join every vertex on the circle to its center with a straight line. Calculate the angles between adjacent lines. If the angles are equal to the symmetry angle within some tolerance, then the vertices are rotationally symmetric about the circle, and hence rotationally skewed symmetric about the ellipse.

4.3.1 Fitting a single ellipse

The first rule for skewed rotational symmetry of a set of vertices in a 2D drawing is:

Rule 1 The vertices must lie on an ellipse.

This rule follows directly from Property 1. It requires checking if a set of given vertices lie on an ellipse. A simple and effective ellipse fitting method was proposed by Fitzgibbon [Fitzgibbon 1999], in which the minimum number of points needed to uniquely determine an ellipse is five. For our purpose, an ellipse needs to be fitted from three points onwards. The cases of three points and more than four points are

discussed in this section. The case of exactly four points is discussed in Section 4.3.3.

Furthermore, if a set of points is rotationally symmetric in real 3D space, Property 2 dictates that they must satisfy:

Rule 2 The vertices must be evenly distributed on the circle in 3D which is projected into the ellipse in 2D.

This means that the centre of the circle must be the mean of the vertices. Given that the fitted ellipse is a parallel projection of this circle, it further implies:

Rule 3 The centre of the ellipse must be the mean of the vertices that fit the ellipse.

Hence, Fitzgibbon's method needs to be modified to have the centre of the ellipse as the mean of the points.

Our derivation of the ellipse follows Fitzgibbon's method, with modification to cater to skewed symmetry. The method does not deal with skewed symmetry of two points because it does not require fitting an ellipse, as two points are always skewed symmetric about their mid point.

The general equation of a conic is

$$F(\mathbf{a}, \mathbf{x}) = \mathbf{a} \cdot \mathbf{x} = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \quad (4.1)$$

where A, B, C, D, E , and F are coefficients of the conic, x and y are the coordinates of the points lying on it, $\mathbf{a} = [A \ B \ C \ D \ E \ F]^T$, and $\mathbf{x} = [x^2 \ xy \ y^2 \ x \ y \ 1]$. $F(\mathbf{a}, \mathbf{x})$ is the "algebraic distance" of the point (x, y) to the conic $F(\mathbf{a}, \mathbf{x}) = 0$.

An ellipse is a special case of a general conic under the constraint $B^2 - 4AC < 0$. Fitzgibbon transformed the inequality constraint into an equality constraint $4AC - B^2 = 1$. The fitting of an ellipse to a set of points (x_i'', y_i'') , $i = 1 \dots N$ may be

done using least square fitting, i.e. by minimizing the sum of the square of the algebraic distances $\mathbf{D}(\mathbf{a}) = \sum_{i=1}^N F(\mathbf{a}, \mathbf{x}_i)^2$, subject to $4AC - B^2 = 1$. Minimization can be solved by considering ranking the deficient generalized eigenvalue system:

$$\mathbf{D}^T \mathbf{D} \mathbf{a} = \lambda \mathbf{C} \mathbf{a} \quad (4.2)$$

where $\mathbf{D} = [\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n]^T$, $\mathbf{C}$ is the matrix that expresses the constraint, which can be expressed in the matrix form as $\mathbf{a}^T \mathbf{C} \mathbf{a} = 1$. The solution of the eigen-system (4.2) gives six eigenvalue-eigenvector pairs. The eigenvector a_k , corresponding to the minimal positive eigenvalue λ_k represents the best-fit ellipse for the given set of points.

If the vertex set is rotationally skewed symmetric, then the center of the fitted ellipse is the mean of the vertices in the given vertex set also. The coordinates of the

center (x_0, y_0) is the mean of the points: $x_0 = \frac{\sum_{i=1}^N x_i''}{N}$, $y_0 = \frac{\sum_{i=1}^N y_i''}{N}$.

Let $\mathbf{x}^* = [(x-x_0)^2 \quad (x-x_0)(y-y_0) \quad (y-y_0)^2 \quad 1] = [x^{*2} \quad x^*y^* \quad y^{*2} \quad 1]$. Then

Equation (4.1) becomes the equation of an ellipse with its centre at the origin

$$F(\mathbf{a}, \mathbf{x}^*) = \mathbf{a} \cdot \mathbf{x}^* = Ax^{*2} + Bx^*y^* + Cy^{*2} + F = 0,$$

where $\mathbf{a} = [A \ B \ C \ F]^T$. At the same time, $\mathbf{C} = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ in (4.2) [Fitzgibbon

1999]. The best-fit ellipse can be obtained by solving (4.2) as well.

The parametric equation of an ellipse centered at $(0, 0)$ is

$$\frac{(x \cos \theta + y \sin \theta)^2}{a^2} + \frac{(x \sin \theta - y \cos \theta)^2}{b^2} = 1$$

where the parameter a is the semimajor radius, b the semiminor radius (assuming $b < a$) and θ the angle between the major axis and the x-axis of the screen coordinate system ($0^\circ \leq \theta < 180^\circ$). Equation (4.1) can be written in the form

$$\frac{a^2 \sin^2 \theta + b^2 \cos^2 \theta}{-2ab} x^2 + \frac{(a^2 - b^2) \sin \theta \cos \theta}{ab} xy + \frac{b^2 \sin^2 \theta + a^2 \cos^2 \theta}{-2ab} y^2 + \frac{ab}{2} = 0 \quad (4.3)$$

From Equations (4.1) and (4.3), we have

$$\begin{cases} A = \frac{a^2 \sin^2 \theta + b^2 \cos^2 \theta}{-2ab} \\ B = \frac{(a^2 - b^2) \sin \theta \cos \theta}{ab} \\ C = \frac{b^2 \sin^2 \theta + a^2 \cos^2 \theta}{-2ab} \\ F = \frac{ab}{2} \end{cases}, \text{ where } 4AC - B^2 = 1.$$

If there are more than four vertices in the vertex set, then after the ellipse fitting, coefficients A , B , C and F are known, so parameters a , b , and θ can be obtained by solving the equations above:

$$\begin{cases} a = \sqrt{F(1 - A - C)} + \sqrt{F(-1 - A - C)} \\ b = \sqrt{F(1 - A - C)} - \sqrt{F(-1 - A - C)} \\ \theta = \arccos\left(\sqrt{\frac{4AF + a^2}{a^2 - b^2}}\right) & B \geq 0 \\ \theta = \arccos\left(-\sqrt{\frac{4AF + a^2}{a^2 - b^2}}\right) & B < 0 \end{cases}$$

When there are only 3 points in the fitting, the parameters A , B , C , and F can be obtained by freely choosing $F = 1$ and solving these equations instead:

$$\begin{cases} Ax_1^{*2} + Bx_1^*y_1^* + Cy_1^{*2} + F = 0 \\ Ax_2^{*2} + Bx_2^*y_2^* + Cy_2^{*2} + F = 0 \\ Ax_3^{*2} + Bx_3^*y_3^* + Cy_3^{*2} + F = 0 \\ F = 1 \end{cases}$$

where (x_i^*, y_i^*) , $i = 1 \dots 3$ are the coordinates of the three vertices. Then we get the parameters of the ellipse fitting by the three vertices:

$$\begin{cases} a^2 = \frac{2}{-A - C - \sqrt{(A - C)^2 + B^2}} \\ b^2 = \frac{2}{-A - C + \sqrt{(A - C)^2 + B^2}} \\ \theta = \arccos \sqrt{\frac{Ab^2 + 1}{Ab^2 + Cb^2 + 1}} & B \geq 0 \\ \theta = \arccos \left(-\sqrt{\frac{Ab^2 + 1}{Ab^2 + Cb^2 + 1}} \right) & B < 0 \end{cases} \quad (4.4)$$

Three points would always fit an ellipse exactly, and thus a unique ellipse will always exist for three points. The ellipse obtained for more than four points is the best fit ellipse, and its actual existence needs to be ascertained by checking that all the points lie on the ellipse. For exactly four points, there is some complication in the ellipse fitting and this is dealt with later in Section 4.3.3.

Freehand drawings are usually inaccurate. To determine rotational symmetry using such inputs, there must be some tolerance for the inaccuracies. For ellipse fitting, tolerance is applied to check that an ellipse actually exists for a given set of points by checking the quality of the fit. As pointed out earlier, for three points, the fitted ellipse will pass through the points exactly. This is not the case for more than four points and therefore a tolerance checking measure needs to be applied.

Figure 4.3: A fitted ellipse and a vertex

Assume that the ellipse has its coordinate system as shown in Figure 4.3. (Normally, we would not get the coordinate axes aligned with the major and minor axes of the ellipse directly from fitting to the given points, but this can be achieved via a simple coordinate transformation.) Let the major and minor diameters of the ellipse be a and b respectively. Let α_i be the angle between the line from the center of the ellipse to (x_i, y_i) and the major axis, l_i be the distance from the center to the same vertex, and r_i be the distance from the center to the ellipse circumference along the line (Figure 4.3), then

$$\begin{aligned}
 l_i^2 &= x_i^2 + y_i^2 \\
 r_i^2 &= \frac{a^2 b^2}{a^2 + (b^2 - a^2) \cos^2 \alpha_i} \\
 \frac{l_i^2}{r_i^2} &= \frac{(x_i^2 + y_i^2) \left(a^2 + (b^2 - a^2) \frac{x_i^2}{x_i^2 + y_i^2} \right)}{a^2 b^2} \\
 &= \frac{a^2 x_i^2 + b^2 y_i^2}{a^2 b^2}
 \end{aligned}$$

We define a tolerance measure, s_e , for how close the N vertices are to the fitted ellipse:

$$s_e = \frac{\sum_{i=1}^N s_i}{N}, \text{ where } s_i = \left| \frac{l_i^2}{r_i^2} - 1 \right|.$$

In our implementation, for a fitted ellipse, we require $l_i/r_i < 0.7$, i.e., $s_e < 0.51$.

4.3.2 Mapping of an ellipse to a circle

Under parallel projection, a circle in 3D is projected into an ellipse in 2D. To determine if a set of 2D vertices distributed on an ellipse are skewed symmetric, we map them on to the 3D circle and check if they satisfy Rule 2, which requires that the points be evenly distributed on the 3D circle.

In Figure 4.4, f denotes the drawing plane and f' denotes the plane of the circle. The intersection line of the two planes is parallel to the major axis, X , of the ellipse and perpendicular to the minor axis, Y . X' is the x axis in the circle and is projected to X , and Y' is the y axis of the circle. X and Y also defines the reference coordinate system in the plane of the ellipse, while X' and Y' defines the corresponding coordinate system in the plane of the circle.

Figure 4.4: A circle and its parallel projection.

In the drawing plane, the semi-major radius a and semi-minor radius b are known after the ellipse fitting. Let β be the angle between f and f' . Because a must be

equal to the radius of the circle in f' and the ellipse is the projection of the circle, $\cos\beta = b/a$. Let α_i be the angle the major axis X makes with the line joining the centre of the ellipse to a point (x_i, y_i) on the circumference (see Figure 4.4). The relationship between α_i and α_i' , the corresponding angle in the plane of the circle, is

$$\begin{aligned}\cos\alpha_i' &= \frac{x_i'}{r'} = \frac{x_i}{a} = \frac{\cos\alpha_i}{a/r_i} \\ &= \frac{\cos\alpha_i}{a/(\frac{a^2 b^2}{a^2 + (b^2 - a^2)\cos^2\alpha_i})} = \frac{\cos\alpha_i}{\sqrt{\cos^2\alpha + \sin^2\alpha / \cos^2\alpha}}\end{aligned}\quad (4.5)$$

where $\cos\alpha_i = \frac{x_i}{\sqrt{x_i^2 + y_i^2}}$.

$$\text{Hence } \begin{cases} \alpha_i' = \arccos \frac{\cos\alpha_i \cos\beta}{\sqrt{\cos^2\alpha_i \cos^2\beta + \sin^2\alpha_i}} & y_i \geq 0 \\ \alpha_i' = \pi + \arccos \frac{\cos\alpha_i \cos\beta}{\sqrt{\cos^2\alpha_i \cos^2\beta + \sin^2\alpha_i}} & y_i < 0 \end{cases}$$

The angles subtended at the centre of the circle by successive point pairs are

given by
$$\begin{cases} \Delta\alpha_0' = \alpha_0' + 2\pi - \alpha_{N-1}' \\ \Delta\alpha_i' = \alpha_i' - \alpha_{i-1}' \quad i = 1 \dots N-1 \end{cases},$$

where N is the number of points. For rotational symmetry, these angles must be the symmetry angle. For fault tolerance, we use a measure s_a , which is the maximum relative deviation among all $\Delta\alpha_i'$ from the symmetry angle $\theta = 2\pi/N$:

$$s_a = \max |\Delta\alpha_i' - \theta| / \theta, \quad i = 0 \dots N-1.$$

In our implementation, we consider the vertices equally arranged on the circle when s_a is less than 0.3, which means the corresponding vertices are equally arranged in the ellipse as well.

4.3.3 Fitting an ellipse to four vertices

This section deals with fitting an ellipse to four points, which is different from the other numbers of points because in certain circumstances, the ellipse fitting four given points, while satisfying Rules 1 and 3, will not be unique. As a reminder, Rule 1 requires the points to lie on an ellipse and Rule 3 requires that the centre of the ellipse be the mean of the points. Figure 4.5 gives an example where the four points are the four corners of a rectangle.

Intuitively, one would think that the three point problem is a special case of the four point problem: an ellipse that fits four points would fit any three of the points as well. This is actually not the case. The ellipse fitted to four points is unique when Rules 1 and 3 are satisfied (see Figure 4.5). By removing one of the points to make it a three point problem, the mean of the points is shifted. Maintaining the same ellipse over the three points would lead to a violation of Rule 3 which requires that the mean of the points to be the centre of the ellipse as well.

Figure 4.5: Three different ellipses that fit four points.

Rules 1 and 3 can be applied jointly to check if four given points are symmetrically distributed on an ellipse. If the distribution is symmetric, then the two

diagonal lines joining the two pairs of opposite points must bisect each other and the bisection point must be the centre of the ellipse. Otherwise, there is no symmetry. If symmetry is established, then we can apply Rule 2 to find the parameters a , b and θ of the fitting ellipse by the method below.

Let P_i , $i = 1 \dots 4$ be the four given points in the sketch and let P_0 be the mean of the four points.

First we take P_0 as the origin of the coordinate system of the ellipse, which means $P_0 = \mathbf{0}$ in this coordinate system. We find the fitting ellipse by finding the ellipse passing through two points P_1^* and P_2^* , where $P_1^* = (P_1 + P_3)/2$ and $P_2^* = (P_2 + P_4)/2$, with P_0 as the centre. Had the four input points been exact, as in the case of the vertices of a rectangle, P_1 and P_1^* would be equal. Given that they are inaccurate, being vertices in a sketch, the two points are not normally equal, but close to each other. The difference is a function of the inaccuracy in the original input. P_1^* , being derived from the average of P_1 and P_3 , averages out the inaccuracies and therefore helps to moderate the errors contained therein. The same relationship exists between P_2 , P_4 and P_2^* .

To find the ellipse that passes through P_1^* and P_2^* with P_0 as the centre, we return to the plane of the circle whose parallel projection is the intended ellipse. Let P_1' , P_2' and P_0' be the points in the circle plane, f' , that correspond to P_1^* and P_2^* with P_0 respectively. They can be found using the method described in the previous section.

The symmetry angle for rotational symmetry with four points is $\pi/2$, which

means the lines $\mathbf{P}_1^* \mathbf{P}_0$ and $\mathbf{P}_2^* \mathbf{P}_0$ should be perpendicular. Let α_1' be the angle $\mathbf{P}_1^* \mathbf{P}_0$ makes with the x-axis in f' , and α_2' the corresponding angle $\mathbf{P}_2^* \mathbf{P}_0$ makes. Then,

$$\alpha_1' - \alpha_2' = \pi/2$$

which means

$$\cos^2 \alpha_1' + \cos^2 \alpha_2' = 1 \quad (4.6)$$

Substituting Equation 4.5 into Equation 4.6 gives

$$\cos \alpha_1 \cos \alpha_2 \cos^2 \beta + \sin \alpha_1 \sin \alpha_2 = 0, \quad (4.7)$$

where β is the angle between the drawing plane and the plane of the circle as defined

in Section 4.3.2. Recall that $\cos \beta = b/a$ and $\cos \alpha_i = \frac{x_i}{\sqrt{x_i^2 + y_i^2}}$.

By substituting (4.4) and (4.5) into (4.7), we then have

$$x_1 x_2 \frac{2}{-A-C+\sqrt{(A-C)^2+B^2}} + y_1 y_2 \frac{2}{-A-C-\sqrt{(A-C)^2+B^2}} = 0$$

where

$$\begin{cases} x_i = x_i^* \cos \theta + y_i^* \sin \theta \\ y_i = x_i^* \sin \theta - y_i^* \cos \theta \end{cases} \quad (i=1, 2),$$

$$\begin{cases} \cos^2 \theta = \frac{Ab^2+1}{Ab^2+Cb^2+1} = \frac{A-C+\sqrt{(A-C)^2+B^2}}{A+C+\sqrt{(A-C)^2+B^2}} \\ \sin^2 \theta = \frac{Cb^2}{Ab^2+Cb^2+1} = \frac{2C}{A+C+\sqrt{(A-C)^2+B^2}} \end{cases}$$

Since we have

$$\begin{cases} x_1 x_2 \frac{2}{-A-C+\sqrt{(A-C)^2+B^2}} + y_1 y_2 \frac{2}{-A-C-\sqrt{(A-C)^2+B^2}} = 0 \\ Ax_1^{*2} + Bx_1^* y_1^* + Cy_1^{*2} + 1 = 0 \\ Ax_2^{*2} + Bx_2^* y_2^* + Cy_2^{*2} + 1 = 0 \end{cases},$$

by solving the equations above, A , B and C of the fitting ellipse can be obtained.

$B^2 - 4AC < 0$ is satisfied too.

4.3.4 The order of rotational symmetry in a polygon

An object in real space has rotational symmetry of order N about an axis if it is invariant under rotation of $2\pi/N$ radians about that axis. An order N symmetry is also referred to as N -fold symmetry. For a polygon with symmetry of order N , the number of vertices in the polygon must be a multiple of N . A symmetric polygon having $q \cdot N$ vertices has q sets of N interlaced vertices. For example, the polygon in Figure 4.6a can be represented as two sets of three interlaced vertices, $\{v_0, v_2, v_4\}$ and $\{v_1, v_3, v_5\}$.

Figure 4.6: Polygons with interlaced subsets of vertices.

For a polygon with n vertices, set N to n/q , $q = 1, 2 \dots$, until its value is less than 2, while admitting only integer values. For each value of N , check if each of the q interlaced sets of N vertices of the polygon can be on the same ellipse and be equally arranged, using the methods established above. If the vertices in the polygon are $v_1 \dots v_n$, then each set is formed by the vertices $\{v_{i+jq}, j = 0 \dots N-1\}$, for $i = 1 \dots q$. There may be different values of N that produce vertex sets that satisfy the rules for symmetry,

then the order of the rotational symmetry of the shape is the largest of the possible values.

For example, for the polygon in Figure 4.6a with six vertices, we start with $q = 1$, and thus $N = 6$, for which there is only one possible vertex set, $\{v_0, v_1, v_2, v_3, v_4, v_5\}$. These vertices clearly cannot lie on the same ellipse; it can be shown computationally that they violate Rule 1. So increase q to 2, and thus $N = 6/2 = 3$. Now, there are two sets of interlaced vertices $\{v_0, v_2, v_4\}$ and $\{v_1, v_3, v_5\}$; both satisfy Rules 1 and 2. So the order of skewed rotational symmetry of the polygon is 3.

The polygon in Figure 4.6b is rotationally symmetric with $N = 6, 3$, and 2 and the required solution is the maximum one, i.e. $N = 6$.

4.3.5 Multiple sets of rotationally symmetric vertices

Multiple sets of rotationally symmetric vertices together may or may not be rotationally symmetric. Further rules are required to determine whether they are. There are four such rules.

Rule 4: If all the sets of vertices belong to the same polygon (and thus the same plane in 3D) or different polygons on the same plane, then all the best-fit ellipses must have the same centre.

This rule is a consequence of Property 3.

Rule 5: If the sets of vertices do not all belong to the same polygon (and thus lie on different planes in 3D), then the centers of all the best-fit ellipses must be collinear.

This rule directly follows from Property 4.

Rule 6: The ratios of the major radius to the minor radius must be the same for all the

$$\text{best-fit ellipses, i.e. } \frac{a_1}{b_1} = \frac{a_2}{b_2} = \dots = \frac{a_m}{b_m}$$

This is a result of Property 5, as the same parallel projection of circles in parallel planes must produce ellipses with the same ratio between the radii of the major and minor axes.

Rule 7: The directions of the axes of all the best-fit ellipses must be the same.

Again, this rule follows from Property 5.

As previously, all the geometric equalities are computed to within a tolerance to allow for inaccuracies.

4.4 Topological analysis

This section describes the procedure to detect a potential axis of symmetry and the faces, edges and vertices topologically symmetric about it. Each symmetry group of vertices is then put through the geometric analysis described above to detect geometric symmetry. First, it is necessary to determine the faces of the object in the drawing.

As in the case for mirror symmetry, we use Liu and Lee's algorithm to identify the faces in the 2D line drawing. See Section 3.4

4.4.1 Topological symmetry detection in a 2D drawing of a 3D model

As mentioned earlier, every face of a rotationally symmetric object must belong

to a collection of faces which are collectively symmetric about the axis; each collection may have one or more members. We call such a collection a *symmetric face set*. The faces in a set must be such that the set is geometrically invariant when rotated about the axis through the symmetry angle. The definitions for *symmetric edge set* and *symmetric vertex set* are similar.

Figure 4.7 shows two rotationally symmetric objects. The object in Figure 4.7a is 3-fold symmetric and has three symmetric face sets: $\{1\}$, $\{2, 4, 6\}$ and $\{3, 5, 7\}$. The object in Figure 4.7b has multiple 2-fold symmetries. For the symmetry axis shown, there are three symmetric face sets each with two faces, $\{1, 3\}$, $\{2, 4\}$ and $\{5, 6\}$. The shadings in the figure highlight the face sets. The symmetric edge and vertex sets are also shown.

Clearly, an object with a face not belonging to any symmetric face set is not rotationally symmetric. Hence our algorithm goes through the graph and attempts to place all the faces in symmetric face sets and, at the same time, place the corresponding edges in symmetric edge sets and vertices in symmetric vertex sets. It quits with no symmetry when it fails to so place a face in a symmetric face set.

(a) Symmetric face sets: {1}, {2, 4, 6}, {3, 5, 7}. Symmetric edge sets: {ab, ad, af}, {ac, ae, ag}, {bc, de, fg}, and {cd, ef, gb}. Symmetric vertex sets: {a}, {b, d, f}, {c, e, g}.

(b) Symmetric face sets: {1, 3}, {2, 4}, {5, 6}. Symmetric edge sets: {bc}, {fe}, {ab, cd}, {bg, ch}, {af, de}, {fg, eh}, {ah, gd}. Symmetric vertex sets: {a, d}, {b, c}, {g, h}, {f, e}

Figure 4.7: Rotationally symmetric objects and their symmetric face sets about the axes shown.

An axis of rotational symmetry of a 3D object must pass through either a vertex or the center of a face, which is the center of the ellipse fitted to the face contour. Additionally, in the case of 2-fold symmetry only, it may also pass through the midpoint of an edge. See Figure 4.8. These are the starting points of the algorithm, from which we can find the symmetry axis of the whole object.

(a) Axis passing through a vertex and the center of a face. (b) Axis passing through two vertices. (c) Axis passing through centers of two faces. (d) Axis passing through two edges in 2-fold symmetry

Figure 4.8: Rotationally symmetric 3D objects and their axes of symmetry.

The following sub-sections provide the details of each step leading towards the

location of a symmetry axis. The first step is to find the seed symmetric edges from which the symmetric face sets, symmetric edge sets and symmetric vertex sets can be found. The symmetric vertex sets are then used to establish the existence of geometric symmetry. The pseudo-code for the algorithm is given at the end of this section.

4.4.1.1 Detecting the seed symmetric edge sets

Seed symmetric edges are edges from which the other symmetric entities may be traced.

When the starting point is the midpoint of an edge, the edge itself becomes a seed edge. When the starting point is the center of a face, it means that the face is rotationally skewed symmetric, and its edges form the seed edges. If the face has N -fold rotational symmetry, then it must have $q \cdot N$ edges split into q sets of N interlaced edges. Each is a set of rotationally symmetric edges. Let l_{ij} be the edge connecting vertex v_i and vertex v_j , in the example of Figure 4.6a. Since $N = 3$, there are two sets of symmetric edges: $\{l_{01}, l_{23}, l_{45}\}$ and $\{l_{50}, l_{12}, l_{34}\}$.

Figure 4.9: Detecting the skewed rotational symmetry beginning from a vertex.

We now look at the case when the starting point is a vertex. Each vertex, V , of an

object is the point of intersection of a set of faces, as well as a set of edges. The other ends of the edges connected to V form a new vertex group, in which the order of the vertices follows the order of their edges around V , which means two adjacent vertices must be in the same face. This is illustrated by the example in Figure 4.9: v_6 is the intersection of l_{06} , l_{16} , l_{26} , l_{36} , l_{46} , and l_{56} , and the new vertex group is $\{v_0, v_1, v_2, v_3, v_4, v_5\}$. Using a method similar to that of finding the order of symmetry in a polygon described in Section 4.3, we can detect the order of symmetry in this vertex group. If the group is symmetric with order N , then it has $q \cdot N$ vertices divided into q sets of N interlaced vertices. The vertices in each symmetric vertex set must satisfy Rules 1 and 2. At the same time, the q sets of N interlaced edges corresponding to the vertices in the vertex group can also be established. One end of an edge in an edge set is V , the other end is in the vertex group. In the example of Figure 4.9, $N = 3$, and there are two set of symmetric edges connected to v_6 : $\{l_{61}, l_{63}, l_{65}\}$ and $\{l_{60}, l_{62}, l_{64}\}$. Rules 4 to 7 establish their symmetry as a group.

4.4.1.2 Detection of symmetric face, edge and vertex sets in the drawing of a 3D model

A symmetry axis of an object must pass through two points, each being the center of a rotationally symmetric face, the mid-point of an edge or a vertex that has rotationally symmetric distribution of connected edges.

Figure 4.10: Sequence of finding face sets

We group the potential axis points—centre of symmetric faces, mid point of edges and existing vertices—into a set, $XP = \{xp_1, xp_2, \dots, xp_n\}$, where n is the number of such points. Take one axis point, xp_i , from XP . The algorithm below checks if xp_i is a point on a rotational axis of the object by analyzing the topological connections of the graph. It identifies, topologically, the symmetric face sets about xp_i , and stores them in a list FS . Similarly, the algorithm also identifies the symmetric edge set, ES and symmetric vertex set, VS . These sets are initialized to null: $FS = \{\emptyset\}$, $ES = \{\emptyset\}$ and $VS = \{\emptyset\}$.

Beginning from xp_i , we identify the seed symmetry edge set as described earlier. If they exist, add them to ES . From each edge set, we can find a new candidate face set whose members are adjacent to the edges in the edge set. For example, in Figure 4.10, let the first edge set be $\{l_1, l_1', l_1'' \dots\}$, where l_1 is adjacent to f_1 , l_1' is adjacent to

f_1', l_1'' is adjacent to f_1'' etc. So $\{f_1, f_1', f_1'' \dots\}$ is a potential symmetric face set if the numbers of vertices of the faces in the set are the same. If the corresponding vertices in the faces of the face set satisfy the rules of Section 4.3, the face set is symmetric and every edge in f_1 is symmetric with the corresponding edges in f_1', f_1'', f_1''' and so on. Then the new detected face set is added to FS and all the new detected symmetric edge sets are added to ES . The vertices of the edge sets are added to VS as well. After that, another new edge set is selected from ES , and its adjacent faces are found and checked again, and the process repeats. In Figure 4.10, let edges $\{l_2, l_2', l_2'' \dots\}$ be one of the new edge sets found in the face set $\{f_1, f_1', f_1'' \dots\}$ respectively, then in the next step, from the new edge set $\{l_2, l_2', l_2'' \dots\}$, a new potential face set $\{f_2, f_2', f_2'' \dots\}$ is detected. The algorithm ends when all the faces adjacent to the edge sets in ES are in FS or when a potential face set is proved to be not symmetric. In the former, the model is symmetric. In the latter, it is not.

Starting from the seed symmetric edge set, the algorithm locates symmetric face sets, from which more symmetric edge sets are found. These new edge sets lead to more face sets. This process terminates early if non-symmetry is found, or it would eventually traverse through every edge and every face in the graph and stop when there are no more new edges sets, and ultimately establish the existence of symmetry.

If x_{p_i} is a point on a symmetric axis, FS , ES and VS will contain the sets of symmetric faces, edges and vertices when the algorithm terminates. The symmetric sets establish the existence of the axis of skewed rotational symmetry. The algorithm started from one point on the axis; the other point can be found using the following

rules:

If the N faces in a face set in FS are the same face, then the center of the face is another end of the axis.

If the N faces in a face set in FS intersect at a vertex, then the vertex is the other point on the axis.

In the case of 2-fold symmetry, if the two faces in a face set in FS intersect at an edge, then the mid point of the edge is the other point on the axis.

Both points on the axis must be in XP . Once the axis has been established, both points are removed from XP . If there is no symmetry, then only the starting point, xp_i , is removed. The process then repeats with another point remaining in XP , until XP is empty.

In summary, our algorithm for detection of skewed rotational symmetry in a 2D line drawing consists of the following steps:

1. Find the skewed symmetric faces and vertices with symmetric connected edges in the drawing. Create the tentative list of points on axes by adding the centers of symmetric faces, the vertices with symmetric distribution of connected edges and mid points of edges to XP .
2. Pop a point, say xp_i , from XP . Set ES , FS and VS to null.
3. From xp_i , find seed symmetric edge sets and add them to ES .
4. Detect symmetric faces from ES and add them to FS .
 - 4.1 If all the pairs in ES have been processed, go to Step 5, otherwise pick a new edge set.

- 4.2 For each edge set, if faces adjacent to the corresponding edges in edge set are not in FS , we check whether the numbers of vertices of the faces are the same and whether the two faces are skewed symmetric.
- 4.3 If the faces in the candidate face set are not symmetric, or the centers of the best-fit ellipses for corresponding vertices in VS are not collinear within some tolerance, then there is no symmetry axis through x_{p_i} . Go to Step 5.
- 4.4 Otherwise, the face set is added to FS , and the corresponding edge sets and vertex sets are added to ES and VS .
5. If x_{p_i} is in an axis of the model, the axis and symmetry sets are saved and the two points on the axis are removed from XP . Otherwise, delete x_{p_i} from XP . Go to Step 2.

4.4.2 Complexity of the algorithm

In the worst case, there are $O(m)$ candidate points for the axis, where m is the number of edges in the drawing. From each point, it needs $O(m)$ time to compute if there exists a skewed symmetry axis containing the point. Therefore the symmetry detection algorithm runs in $O(m^2)$ time. Practical cases are expected to be substantially better than the worst case since, in general, the number of symmetries in an object is a lot less than the number of edges, which means that in the majority of cases, the search for a symmetry axis will quit early (see Step 4.3 in the algorithm). The experiments described in Figure 4.11 show that the computation time is reasonable.

4.5 Results

Figure 4.11 shows some results of detecting skewed rotational symmetry planes on a variety of drawings representing objects of various types. As seen in the examples, the algorithm can detect skewed rotational symmetry correctly. The dashed lines are the symmetry axes detected. Axes of 2-fold symmetry are not shown because there can be many of them and they can clutter up the picture. The computational times for most of the examples are less than 0.005 sec using a Pentium 4 PC.

The algorithm finds the symmetry axis from the points of intersection it makes with the object. The point set XP contains all the possible points of intersection between an axis and the boundary of the object. The algorithm uses these points as starting points to look for the axes, and would therefore find all the axes that intersect the object. Hence it is possible that axes that do not intersect the object would not be found.

Figure 4.12a shows such an object, where the vertical axis of symmetry does not intersect any face of the object. The fix to this problem is to modify the algorithm so that it takes into account centers of symmetric hole loops. This requires an enhancement to the face detection algorithm to return the face loops and the hole loops.

Figure 4.11: Examples

Figure 4.12: Two objects with axes through holes.

Note, however, that this problem only exists when the hole loop is not contained in a face of the object, which is the case for the object in Figure 4.12a. In the case shown in Figure 4.12b, where the two hole loops are contained within two faces of the object, the peripheral loops of the faces provide the means to identify the points on the axis.

4.6 Conclusions

A new algorithm for the detection of the skewed rotational symmetry axes of a line drawing of a 3D polyhedral model has been presented. It works by identifying topologically symmetric sets of vertices and checks that they are geometrically symmetric by ensuring that they satisfy the rules given in Section 4.3. We have implemented the algorithm and the tests show that it works well, with adjustable fault tolerance.

The algorithm can be improved by relaxing some of the restrictions stated in Section 4.3, such as the handling of drawings with hidden lines removed. Missing

edges mean that some information required by our algorithm to derive the symmetry axes is not available, and needs to be filled in automatically. Such filling in can only be speculative, but this can be aided by checking the symmetry of the visible entities. A face containing a point through which a symmetry axis passes must be rotationally symmetric itself. Likewise, if this point is a vertex or the mid point of an edge, then the faces bordering this vertex or edge must be collectively rotationally symmetric. It is possible to use such a point as the starting point to attempt to construct the missing symmetric faces.

CHAPTER 5

Detecting Minimum Over-constrained Subgraphs in 2D and 3D Based on Degree of Freedom Analysis

A flow-based method is presented for finding minimum over-constrained subgraphs in a geometric constraint graph. In 2D, several different approaches have been implemented, while the 3D problem, in which the entities are points, lines and planes of a model, is much less investigated in the literature. In this chapter, a general algorithm is presented for the problem both in 2D and 3D and show that the algorithm is correct and can be used to determine whether a structurally over-constrained subgraph exists in a constraint graph.

5.1 Introduction

One method of beautification is to set up the constraints between the entities, and resolve the dimensions based on these constraints computationally. Some of the constraints can be established automatically, but there will always be some that must be given by the user. Clearly, it is desirable to minimize the user input on such a potentially tedious and error-prone task.

Whether assigned manually or automatically, it is necessary to determine if a set of constraints is sufficient for describing a model completely. There can be overconstraint or underconstraint, and there may also be redundancies, which include structural and numerical redundancies. A structural redundancy over-constrains the system. For instance, the constraint $f(x_1, x_2) = 0$ which constrains two variables x_1 and

x_2 in a system will lead to a structural redundancy if two other constraints $g_1(x_1, x_2) = 0$ and $g_2(x_1, x_2) = 0$ are present, since the values of x_1 and x_2 are implicitly determined already by g_1 and g_2 . Constraints and the entities they constrain can be represented as a graph, and f , g_1 , and g_2 result in an over-constrained subgraph. The problem can be rectified by discarding one of the constraints. A system can be numerically redundant or inconsistent. For example, two constraints expressed by $x_1 + x_2 = 1$ and $x_1 + x_2 = 0$ are inconsistent; $x_1 + x_2 = 1$ and $2x_1 + 2x_2 = 2$ are redundant. In this project, a large number of constraints are detected first; the details of how to detect constraints from a 2D sketch and a 3D reconstructed model will be described in Section 6.2.3. Some detected constraints may be redundant or inconsistent, so a method to identify them.

A constraint system can be expressed as a system of equations [Light 1982]. If structural redundancy exists in the system, then the Jacobian matrix of the system of equations is singular. Light [Light 1982] identified an invalid dimensioning scheme by singularity of the Jacobian matrix. Buchanan [Buchanan 1998] determined whether a system of equations is inconsistent by using the Grobner basis [Buchberger 1985]. Gao [Gao 1998a] gave a complete method for deciding whether a set of constraints is independent and whether the constraint system is numerically inconsistent based on Wu-Ritt's decomposition algorithm [Wu 1994]. Despite the advantages of the algebraic approaches using Grobner bases or the Wu-Ritt method, both of them involve exponential time complexity. It is not uncommon for them to take some tens of minutes or even hours, which is not acceptable in a real time interactive system.

Numerous works have addressed the problem of structural redundancies. In 2D,

the triangle decomposition method was used by many researchers in geometric constraint solving [Owen 1991; Bouma 1995; Fudos 1998; Gao 1998a; Joan-Arinyo 2003], so their methods for identifying over-constrained subgraphs were based on triangle decomposition too. For example, in Fudos's method [Fudos 1996, 1997; Fudos 1998], if two well-constrained clusters share more than one geometric element, then over-constraint is detected. But the method can be used only within a limited domain, and cannot be applied in 3D.

Latham [Latham 1996] proposed a method based on the maximum b-matching algorithm to decompose a constraint problem into a sequence of solvable subgraphs. The crux of this algorithm is the detection and correction of over-constraints and under-constraints. But this method of detection does not work well in some cases. For example in the case depicted in Figure 5.2(a), this method cannot detect the over-constrained subgraph $p_2p_3p_4p_5$. Maximum b-matching is a special case in generalized maximum matching (MM), as Hoffmann stated [Hoffmann 2001a], "the MM method may or may not detect over-constrained subgraphs, depending on the initial choice of points for reducing weights". He proposed a method MM1 to correct this drawback, but did not deal with 3D cases.

Li's method can detect over-constraint in 3D cases [Li 2002], but all the entities in the constraint graph must have six degrees of freedom. So his algorithm cannot be used in cases where the entities are points, lines and planes, which do not have six degrees of freedom. Recently, Langbein [Langbein 2004] proposed a method of identifying over-constraint based on Kramer's degree of freedom analysis [Kramer

1992] and Li's method of analyzing dependencies between geometric objects. But his method did not have a rigorous proof, and he could not obtain the minimum over-constrained subgraph. Jermann proposed a new concept of extensive structure rigidity [Jermann 2002; Jermann 2003], which is useful and will be used in our algorithm. His algorithm may find all the over-rigid subgraphs, but the number of sub-objects (called DOR-minimals in the paper) the algorithm has to deal with is large, even though it may stop early without utilising all the sub-objects. Jermann's paper shows that Hoffmann's algorithm for identifying dense subgraphs may fail because of geometric rigidity [Hoffmann 1997b].

Deciding whether a constraint system contains structurally redundant constraints and identifying over-constrained subgraphs are key problems in geometric constraint-solving. This chapter presents a new algorithm to find the minimal over-constrained subgraph in a constraint system and identify the existence of structural redundancy, thus allowing the user to take action to keep the system properly constrained. The rest of this chapter concentrates on structural properties of a constraint system and ignores numerical redundancy.

5.2 Finding over-constrained subgraphs

Finding over-constrained subgraphs in a constraint system is concerned with the association between geometric elements and their constraints. In this thesis, the analysis is presented mainly in the 3D domain; 2D cases are simpler. This section presents the algorithms for identifying over-constrained subgraphs. Before that, there

are some essential fundamental definitions.

5.2.1 Definitions and graphical representation

Figure 5.1: A 3D example.

A **geometric constraint system** can be represented by a constraint graph $G = (V, E)$, where V is the set of nodes representing the entities, including points, lines and planes. E is the set of arcs representing constraints between the entities. For example, Figure 5.1(b) is the constraint graph of the constraint system shown in Figure 5.1(a). The V of its constraint graph is composed of two planes, f_1 and f_2 , two points, p_1 and p_2 , and one line, e_1 . The E of the constraint graph is composed of parallel and distance constraints between f_1 and f_2 , the angle between e_1 and f_1 , and the distance between p_1 and p_2 , with p_1 lying on f_1 , p_2 lying on f_2 , p_1 lying on e_1 and p_2 lying on e_1 .

Finding the over-constrained subgraph is a case of analyzing the degrees of freedom in the system.

The **weight of an entity** is equal to the degrees of freedom (DOF) of the entity.

As shown in Table 5.1, in 3D, the weight of a point is three, for the three translations in each of the three spatial dimensions x, y and z; i.e. the point is free to move in 3D. The weight of a line is four, for two translations along two axes orthonormal to the line, and two rotations about these two orthonormal axes. The weight of a plane is three, for one translation along the plane normal and two rotations about the axes orthonormal to the plane normal.

Entity	Local coordinate system	Freedom of translation	Freedom of rotation
point		x axis y axis z axis	none
line		y axis z axis	y axis z axis
plane		z axis	x axis y axis

Table 5.1: Freedoms of geometric entities in 3D.

The **weight of a constraint** is the number of DOF eliminated by the constraint, which is the number of equations required to define that constraint. For example, the constraint of distance between two points requires one equation that eliminates one degree of freedom, thus its weight is 1.

The relationships between different pairs of basic geometric entities, the constraints between them and the degrees of freedom these constraints consume in 3D are listed in Table 5.2.

Entity 1	Entity 2	Constraint type	Constraint weight	DOR of rigid graph
point	point	distance between two points	1	5
point	line	distance between point and line	1	6
		point lies on line	2	5
point	plane	distance between point and plane	1	5
		point lies on plane	1	5
line	line	angle between two intersecting lines	2	6
		parallel & distance between two lines	3	5
line	plane	parallel & distance between line and plane	2	5
		Angle between line and plane	1	6
		line lies on plane	2	5
		perpendicular between line and plane	2	5
plane	plane	Angle between two planes	1	5
		parallel & distance between two planes	3	3

Table 5.2: Basic constraint types.

The **DOF of a graph** G is equal to the difference between the sum of the weights of the entities in the graph and the sum of the weights of its constraints.

A **rigid subgraph** is a well-constrained subgraph, in which the relative positions of the entities are fixed. In Table 5.2, the entity-entity pair plus the constraint in each row forms a rigid graph. The **degree of rigidity (DOR) of a subgraph** is the number of independent displacements it admits, i.e. the DOF of the corresponding rigid subgraph. Hence the DOR of a rigid subgraph is the same as its DOF. DOR depends on the geometric properties of the entities and the constraints [Jermann 2003]. For example, the DOR of a line (a pair of points and the distance between them, which form a rigid subgraph) is 5 in 3D, and the DOR of two parallel planes in 3D is 3.

In the case of 2D planar constraint problems, the entities are points and lines. The weights of all the entities are two, and all the constraints one.

In Figure 5.2(a), the constraint graph is composed of five 2D points and seven

point-point distances identified by the lines between them. Subgraph $p_2p_3p_4$ is rigid (see Figure 5.2 (b)), and its DOF is 3, so the DOR of subgraph $p_2p_3p_4$ is 3. In Figure 5.2(c), though the DOF of $p_2p_3p_4p_5$ is 2, the DOR of subgraph $p_2p_3p_4p_5$ is 3, because the DOF of its corresponding rigid subgraph $p_2'p_3'p_4'p_5'$ is 3. In Figure 5.2(d), the DOR of subgraph $p_1p_2p_5$ is 3 though its DOF is 4.

Figure 5.2: A 2D example of degree of rigidity.

If the DOF of a subgraph is less than its DOR, the subgraph is **over-constrained**. In Table 5.2, all the subgraphs consisting of two entities and one constraint are rigid, since the relative positions between the two entities cannot be changed, i.e. the subgraphs are not deformable.

5.2.2 Property of rigid subgraphs

Property of DOR computation [Jermann 2003]: if O_1 and O_2 are two rigid subgraphs, and $O_1 \subset O_2$, then $DOR(O_1) \leq DOR(O_2)$. Because DOR represents the relative displacements allowed in a subgraph, and adding objects to it cannot remove the displacements, it means DOR may increase but cannot decrease with increasing number of objects.

The section below analyses the DOF of different rigid subgraphs. The results will be applied to the algorithm of identifying over-constrained subgraphs.

Theorem 1 In 3D cases, the DOR of any nontrivial subgraph is 3, 5 or 6.

Proof: A graph with only one vertex and no edges is called a trivial graph. Table 5.2 lists the types of constraint between two basic entities in 3D, and all of them are rigid. The DOF of the graphs, which consist of two entities and one constraint in the table are 3, 5 or 6. By adding entities and constraints to the basic graphs in Table 5.2, we can obtain new rigid graphs. If the basic graph contains two parallel planes and the distance between them, then the DOF of the rigid graph is 3. After adding another parallel plane and its distance from one of the existing planes, the DOF of the new rigid graph remains at 3, because in the new rigid graph the sum of the weights of the three nodes is 9 and the sum of the weights of the two constraints is 6 (parallel and distance constraints between two planes). So this gives a DOF of 3 to the new rigid graph. This is the only case where the DOF of a new rigid graph is 3. If we add other types of entity and constraint to a graph with a DOF of 3 to form a new rigid graph, the new graph then contains other types of subgraphs, whose DOR may be 5 or 6 (see Table 5.2). According to the property of DOR computation, the new graph's DOR is 5 or 6.

When the DOF of the basic graph in Table 5.2 is 5, after adding entities and constraints to the basic graph to obtain a new rigid graph, according to the

property of DOR computation, we can only obtain a new rigid graph with $\text{DOR} \geq 5$, which means either 5 or 6.

After adding entities and constraints to a basic graph with $\text{DOF} = 6$, the new rigid graph's DOR may be ≥ 6 , according to property of DOR computation. However, the maximum number of displacements possible in a valid 3D model (i.e. a rigid graph) is 6. So the new rigid graph's DOR has to be 6.

It is clear that the DOR of any rigid subgraph is 3, 5, or 6 from the analysis above. ■

It is easy to identify a rigid subgraph with $\text{DOR} = 3$ in a constraint graph. The subgraph is a set of parallel planes with known distances between them. We will focus on identifying rigid subgraphs with $\text{DOR} = 5$ in a constraint graph by analyzing the properties of its entities and constraints. There are four different types of rigid subgraph with $\text{DOF} = 5$; they are discussed below.

1. For a rigid subgraph with $\text{DOF} = 5$ containing a point and a line, the point must lie on the line.

As stated before, a point has $\text{DOF} = 3$ and a line has $\text{DOF} = 4$. A non-rigid graph containing a line and a point unconstrained will have $\text{DOF} = 7$. If the distance between the line and the point is given, the system becomes a rigid graph with $\text{DOF} = 6$. For the DOF to be 5, we need to remove the rotational freedom of the point about the line by requiring that the point lies on the line.

2. For a rigid subgraph containing one point, one plane and the perpendicular

distance from the point to the plane, the $\text{DOF} = 5$. In the case of multiple points and multiple planes, for $\text{DOF} = 5$, all the planes must be parallel and the points must lie in one line perpendicular to the plane.

A plane has $\text{DOF} = 3$ and a point also has $\text{DOF} = 3$. An unconstrained system with a plane and a point has $\text{DOF} = 6$. Specifying the distance between the plane and the point reduces the DOF by 1, hence such a system has $\text{DOF} = 5$. Adding another point to the system increases its DOF to $5+3 = 8$. Fixing the distance between this new point and the plane reduces the DOF to 7. Further requiring that this point lies on the plane normal containing the first point reduces the DOF by another 2 (the distance between the two points and the rotation of the second point about the said normal). Hence, for a system with a plane and two points, the $\text{DOF} = 5$ when the points lie in the same normal to the plane. The same is true when there are more than two points. Further, it can be shown that if there are multiple planes, then they must be parallel to maintain $\text{DOF} = 5$, since the DOF of a system of parallel planes with known distances between them is 3.

3. For a rigid subgraph with $\text{DOF} = 5$ containing a line and a plane, the line must be perpendicular to the plane or parallel and at a given distance to the plane.

A line has $\text{DOF} = 4$, and a plane has $\text{DOF} = 3$; thus an unconstrained system with these two entities has $\text{DOF} = 7$. Constraining the line to be perpendicular to the plane removes two freedoms: the rotation of the line about the plane normal through their point of intersection, and the translation of the line along this normal. Hence the DOF is reduced to 5. In the parallel case, the line being parallel to the plane and the

distance between them reduce the DOF by 2, hence the DOF of the resulting constraint system is also 5.

4. Except for the cases listed in Table 5.2, there are no other rigid subgraphs with $\text{DOF} = 5$ which have only points or lines or planes.

Based on the discussion above, we can find a rigid subgraph with $\text{DOF} = 5$ by the types of entity and constraint in a given graph. The case of $\text{DOF} = 6$ will not be discussed here as it will be seen in the algorithms below that there is no need to deal with such cases directly.

Given a set of constraints, if there is only one minimum over-constrained subgraph or several minimum over-constrained subgraphs without overlapping of any constraints, then the result is independent of the order of adding the constraints. If the minimum over-constrained subgraphs have some common constraints, then the different order of adding constraints may produce different results. How to order the constraints added is discussed in Section 6.2.3

5.2.3 Algorithm for finding a minimum over-constrained subgraph

The algorithm starts with a subgraph G' of G ; initially G' includes only all the entities and no constraint. Therefore, we know G' is under-constrained, and there is no over-constrained subgraph in G' . We then add the constraints of G to G' one by one. When a constraint e is added to G' , we attempt to find the over-constrained subgraph, starting from the two end points, a and b , of e .

Figure 5.3: Variables in the algorithm.

When the weight of $e = 1$, if there is no rigid subgraph with $\text{DOF} = 5$ or 6 in G' (which does not contain e) including the two end points, then no over-constrained subgraph is found. Otherwise a subgraph G'' would be identified by Algorithm 5.2, and $\{G'', e\}$ is the over-constrained subgraph. The DOF of G'' may be 5 or 6 when the weight of e is 1 , according to Theorem 1. Figure 5.3 shows a sketch of the G' , G'' , base graph and a, b . The base graph begins with $\{a, b\}$; some of the elements of G may be added into it in Algorithm 5.1, then $\{\text{base graph}, e\}$ represents an acceptable rigid subgraph whose DOF maybe 3 or 5 . When the weight of $e = 2$, Algorithm 5.2 will identify a subgraph G'' with DOF of $5, 6$ or 7 , if it exists. And when the weight of $e = 3$, we need to find a subgraph G'' with DOF less than 8 .

It is easy to detect a subgraph with weight 3 , as only a series of parallel planes can have $\text{DOF} = 3$. We can detect such a situation by searching the groups of parallel planes, two of which are end points of e .

Algorithm 5.1: Finding a minimum over-constrained subgraph

```

1.  $G' = \{V\}$ 
2. for every constraint  $e$  in  $G$  do
3.    $base\ graph = e$  and  $a, b$  which are two end nodes connected by  $e$  in  $G'$ 
4.    $G'' = \{\emptyset\}$ 
5.   if can identify a rigid subgraph  $G''$  with  $DOF = 5$  including  $base\ graph$ 
6.     an over-constrained subgraph  $\{G'', e\}$  is found, break
7.   if can identify a subgraph  $G''$  with  $DOF = 6$  including  $base\ graph$ 
8.     if weight of  $e$  is 1
9.       if  $\{G'', e\}$  is one case of rigid subgraphs with  $DOF = 5$ 
10.        add  $G''$  to  $base\ graph$ , go to line 7
11.      else an over-constrained subgraph  $\{G'', e\}$  is found, break
12.   if can identify a subgraph  $G''$  with  $DOF = 7$  including  $base\ graph$ 
13.     if weight of  $e$  is 2
14.       if  $\{G'', e\}$  is one case of rigid subgraphs with  $DOF = 5$ 
15.        add  $G''$  to  $base\ graph$ , go to line 7
16.       else an over-constrained subgraph  $\{G'', e\}$  is found, break
17.   if can identify a subgraph  $G''$  with  $DOF = 8$  including  $base\ graph$ 
18.     if weight of  $e$  is 3
19.       if  $\{G'', e\}$  is one case of rigid subgraphs of  $DOF = 5$ 
20.        add  $G''$  to  $base\ graph$ , go to line 7
21.       else an over-constrained subgraph  $\{G'', e\}$  is found, break
22.   no over-constrained subgraph exists, add constraint  $e$  to  $G'$ 
23. endfor

```

Algorithm 5.2, for identifying a subgraph with $DOF = n$, is based on the “maximum flow” algorithm of Ford-Fulkerson [Gondran 1982]. For a given graph G' , by distributing the weights of the constraints to the DOF of the entities, we can convert the structure graph $G' \{V, E\}$ to a directed graph.

A new constraint network graph (s, V^*, t, E^*, w) is constructed for finding the direction of the arcs in a constraint graph $G(V, E)$. The points in V^* include the points in V and the arcs in E . s is the source, connected to every constraint in E . The capacity of the arc (s, v) , $v \in E$ is equal to the weight of v in E . t is the sink connected to every node in V . The capacity of the arc (v, t) , $v \in V$ is equal to the weight of v in V . The

capacity of the network arc (v_1, v_2) , $v_1 \in E$, $v_2 \in V$, v_2 being constrained by v_1 , is infinite. The arcs in E^* include the arcs between s and the points in V^* , between t and the points in V^* , and the arcs between the points in V^* . w is the capacity of the arcs in E^* . For example, Figure 5.4(b) shows the associated network graph of the object in Figure 5.4(a). The source can be viewed as a producer of some sort of fluid, and the sink as a consumer of the fluid. The maximum flow algorithm determines the largest possible amount of flow that can be sent from the source to the sink along each of the arcs in the constraint network graph. This maximum flow enables the conversion of the constraint graph into a directed graph via flow operations. In a network graph, for every arc (e, v) , $e \in E$, $v \in V$, v is constrained by e . If the flow of e is not equal to zero, then the direction of e is directed to v in the constraint graph.

In a graph with $\text{DOF} = n$, after adding n extra constraints to the graph, the weight of each entity is equal to the sum of the weights of the constraints directed to it. All the weights of the constraints are distributed to the entities connected by the constraints. If the weight of one constraint is increased by 1 in a graph with $\text{DOF} = n$, after distributing all the constraints, the flow going out from the source is less than the weight of the constraint, i.e. one of the constraints cannot be totally absorbed by the entities.

Before adding the last constraint, e , G' is under- or well- constrained. To identify a subgraph with $\text{DOF} = n$ including a , b , which are the two nodes connected by e , Algorithm 5.2 proceeds as follows: First, n extra constraints are added to the nodes a , b , where n is the DOF of the subgraph, n may be 5, 6, 7 or 8. All the constraints in E' ,

which consists of the constraints connecting the entities in the *base graph* and other entities in G' , are deleted. Then the constraints in E' are added one by one to G' . For every constraint in E' , the weight of the constraint is increased by one to check the existence of a subgraph with $\text{DOF} = n$ in G' . By distributing all the constraints, if one weight of a constraint cannot be absorbed by the entities, then a subgraph with $\text{DOF} = n$ exists. Now by doing a breadth-first search, beginning from the last added constraint in E' , we can identify a set of entities, in which the weight of every entity is equal to the sum of the weights of the constraints distributed to the entity. Then a subgraph G'' can be found and identified by the set of entities. G'' is a subgraph with $\text{DOF} = n$ if we add n extra constraints on nodes a and b , so DOF of G'' is n .

When a new constraint is added in line 7 of Algorithm 5.2, the new maximum flow can be found more easily, because we only need to find the new argument path [Gondran 1982] for the newly added constraint.

The technique here is similar to the one used by Hoffmann [Hoffmann 1997b], but Hoffmann distributed arcs connected to an entity, whereas our method distributes arcs connect to the base graph. In 3D cases, when the entities are points, lines and planes, Hoffmann's method [Hoffmann 1997b] will stop at identifying every basic constraint type listed in Table 5.2 as dense subgraphs, in which the difference between the sum of the weights of the entities and the sum of the weights of the constraints is less than 6. A dense graph maybe a well-constrained or over-constrained subgraph, so his method cannot identify a subgraph with certain DOF .

Algorithm 5.2: Identifying subgraph with $\text{DOF} = n$

Input (G' , *base graph*, n), where n is 5, 6, 7 or 8

1. $\text{int } k = 0$
2. set the weights of all the arcs in *base graph* to 0, and weights of all the nodes except for a, b to zero
3. add n extra constraints to the nodes a, b
4. delete all the constraints in E' which connect the entities in *base graph* and other entities in G'
5. **for** every constraint e' in E'
6. add e' to G'
7. increase the weight of e' by 1
8. use maximum flow method to distribute all the constraints to the entities.
 The constraints include constraints in G' , n extra constraints and e'
9. **if** one weight of a constraint cannot be distributed
10. $\{ k = k+1; \text{obtain a subgraph } G_k'' \}$
11. decrease weight of e' by 1
12. **endfor**
13. **if** $k > 0$ **return** G''
14. **else return** no subgraph with $\text{DOF} = n$

The two theorems below prove that an over-constrained subgraph $\{G'', e\}$ is a minimal over-constrained subgraph in G , and it is correct.

Theorem 2 The over-constrained subgraph in Algorithm 5.1 contains the last added constraint e .

Proof: Let A be an over-constrained subgraph that does not contain the constraint e .

Then there should be an arc that was in an over-constrained subgraph before e was considered. This contradicts the assumption that all arcs in G' have not been in an over-constrained subgraph. ■

Theorem 3 The over-constrained subgraph $\{G'', e\}$ detected in Algorithm 5.1 is the minimal over-constrained subgraph.

Proof: If there is a smaller over-constrained subgraph G^* inside G'' , for all the constraints connecting the nodes between G'' and G^* , the directions of the constraints are from G^* to G'' , then it is impossible to find G'' after G^* , so $\{G'', e\}$ is a minimal over-constrained subgraph.

Assume there is another minimal over-constrained subgraph which intersects the detected one. If the intersection contains more than $\{a, b\}$, then $\{G'', e\}$ cannot be found as a minimal over-constrained subgraph. This can be proved by the method stated in the last paragraph.

If the intersection is $\{a, b\}$, Algorithm 5.2 can detect more than one subgraph G_k'' . This special case is not dealt with in Algorithm 5.1, to keep it simple and easy to follow; we describe it here instead. In line 19 of Algorithm 5.1, if more than two of $\{G_i'', e\}$ ($i = 0, 1 \dots k$) are detected as over-constrained subgraphs, the minimal over-constrained subgraph in Algorithm 5.1 is then set to $\{e\}$ in this case. We will simply delete constraint e from the constraint graph to keep it from becoming over-constrained. Theorem 3 requires that the over-constrained subgraph G'' is a minimal over-constrained subgraph. It is not possible, before constructing G'' , for some other part of G (which may share no constraints with G'') to be over-constrained. This is because this over-constrain was eliminated already by the deletion of some of the constraints in the former minimal

over-constrained subgraph before constructing G ■

In 2D, the entities are points and lines, and the weights of both of them are 2, and the weight of a constraint is 1; the DOF of all the rigid subgraphs in 2D are 3. So in the algorithm for finding a minimum over-constrained subgraph, for each e in G' , we need only to find if there exists a rigid subgraph with $\text{DOF} = 3$ including the two end points of constraint e . If such a rigid subgraph exists, then G'' is returned from Algorithm 5.2.

5.3 Complexity analysis

The complexity of Algorithm 5.2 is dominated by that of the Ford-Fulkerson algorithm, which is $O(mn(n+m/2))$, where n is the number of nodes and m is the number of constraints in a constraint graph. It can be shown that that on average, there are $4m/n$ constraints connected to a base graph, so the complexity of Algorithm 5.2 is $O(m^2(n+m/2))$. Then, the complexity of Algorithm 5.1 is $O(m^3(n+m/2))$.

5.4 Example of application

In this section, we show three examples on how the algorithm of this chapter can be used to identify the over-constrained subgraphs in a constraint graph.

5.4.1 A simple 2D example

In Figure 5.4, the constraint graph consists of five 2D points and seven point-point distances identified by the lines between them.

According to Latham's method [Latham 1996], if we add two extra constraints to

point p_1 , and one extra constraint to point p_2 , shown in Figure 5.4(c), then after the computation of maximal flow, all the flow from the constraints go to the entities and there will be no over-constraint or under-constraint in the example.. But over- and under-constraint exist in the example: $p_2p_3p_4p_5$ is an over-constrained subgraph, and line p_1p_2 is under-constrained, as it can rotate around p_2 .

Our algorithm begins with G' which contains all the entities, then the constraints are added one by one to G' . If the distance constraint between p_2 and p_4 is the last constraint added, by distributing the DOF of the constraints to the DOF of the entities, a directed graph is obtained, as shown in Figure 5.4(d). The minimum over-constrained subgraph $p_1p_2p_3p_4$ can be deduced from it, because the graph G'' $p_1p_2p_3p_4$ without the distance between p_2 and p_4 is a well-constrained subgraph with $\text{DOF} = 3$, which is identified by Algorithm 5.2. We can make $p_1p_2p_3p_4$ well constrained by deleting any constraint inside it or the distance constraint between p_2 and p_4 . After deleting one constraint, the point p_1 is still unsaturated, so the under-constraint still exists in the constraint graph.

Figure 5.4: A 2D example on finding minimum over-constrained subgraph.

5.4.2 A simple 3D example

Figure 5.5 shows an example in 3D. The V of its constraint graph is composed of two planes, f_1 and f_2 , two points, p_1 and p_2 , and one line, e_1 . The E of the constraint graph is composed of parallel and distance constraints between f_1 and f_2 , the angle between e_1 and f_1 , the distance between p_1 and p_2 , with p_1 lying on f_1 , p_2 lying on f_2 , p_1 lying on e_1 and p_2 lying on e_1 .

Figure 5.5: A 3D example.

At the beginning, G' contains all the entities only, then the constraints are added to it one by one. If the angle constraint between e_1 and f_1 is the last constraint to be added, then Algorithm 5.1 proceeds as follows: firstly, identify rigid subgraphs with

DOF = 5 by calling Algorithm 5.2, which adds 5 extra constraints to e_1 and f_1 . No rigid subgraph with DOF = 5 is found. Next Algorithm 5.1 looks for subgraphs with DOF = 6. Algorithm 5.2 similarly adds 6 extra constraints to e_1 and f_1 , and a subgraph G'' , $f_1 f_2 p_2 e_1 p_1$ is found. The DOF of the subgraph is 6, as shown in Figure 5.5(d). Then, the minimum over-constrained subgraph is identified (see Figure 5.5(c)), which includes $f_1 f_2 p_2 e_1 p_1$ and the constraint between f_1 and e_1 . So, one of the constraints in the minimal over-constrained subgraph should be deleted. If we add the constraints into G' from the beginning in a different sequence, the same over-constrained subgraph will be identified too.

5.4.3 Another 3D example

Figure 5.6 gives another example using a 3D pyramid. This pyramid contains four points, six lines, four planes and six distance constraints, each with a weight of value 1 each, which are the distances between the points. In the pyramid, the four points are each constrained to lie on three planes, and therefore there are 12 constraints of point on plane, each with a weight of 1. Similarly, there are 12 constraints of point on line, each with a weight of 2. If a user tries to add another constraint indicated by the dashed line in Figure 5.6(b), say the angle between two planes f_1 and f_2 , the graph would be over-constrained, then a minimal over-rigid subgraph would be detected (see Figure 5.6(c)). The minimal over-rigid subgraph includes all the points, the distances between them, f_1 , f_2 and the angle constraint. Then we get a well-constrained system after deleting one constraint in the minimal over-constrained subgraph.

Figure 5.6: A 3D example using a pyramid.

The examples in this section are necessarily simple, because otherwise the constraint diagrams become too complex and unwieldy to show here. Nevertheless, they illustrate the way the algorithms work. More 3D examples with timings for detecting over-constrained subgraph will be given in Section 6.5.

5.5 Conclusions

This chapter describes a flow-based method for identifying over-constrained subgraphs in a constraint system. Basically, it exploits the degree of freedom and degree of rigidity properties of subgraphs, and its method of adding the constraints one by one to the entities in the graph allows all the over-constrained subgraphs to be identified and corrected as the adding progresses.

The algorithm has been implemented in our test system. Experiments show that the algorithm can identify over-constraint both in 2D and 3D, which corrects the mistake of Latham's algorithm, and can handle 3D constraint problems where entities are points, lines and planes.

CHAPTER 6

Beautification and Dimensioning of the 3D Model Reconstructed from a 2D Sketch

3D models reconstructed from 2D sketches are inaccurate because of the inherent inaccuracies in the input and the reconstruction methods. It is therefore necessary to beautify them before use in CAD systems. We present a method that detects geometric constraints, such as parallel and orthogonal faces, present in the reconstructed model and then selects a subset. The constraints in the subset need to be sufficient and consistent. The previous chapter described a graph-based method to detect redundant or inconsistent constraint. This chapter deals with a new approach, based on quasi-Newton optimization, to detect and eliminate redundant constraints. The remaining consistent constraints are then used to define completely the dimensions of the model. Experiments show that the method can beautify and dimension recovered 3D models correctly.

Symmetry has not been included as a constraint in the study. This is because symmetry serves only to reduce the size of the problem by reducing the number of entities involved; the basic algorithm would remain unchanged. Further elaboration is given in Section 6.6.

6.1 Introduction

A 3D model as perceived in human minds is recovered from a sketch, but it is

inaccurate because the sketch, by its very own nature, is inaccurate. The recovery method also affects the accuracy of the model. For example, the vertices in one face may not lie exactly on one plane and parallel faces may not be exactly parallel. Also, the size of the recovered model usually bears no resemblance to the actual size the designer has in mind, as a 2D sketch carries no dimensions. Improvement on the recovered model is therefore necessary to enforce proper geometric relationships and give it proper dimensions before the model can be used further. Figure 6.1 illustrates the main stages.

Figure 6.1: Conceptual design by sketching.

It is neither practical nor desirable to require the user to provide the data for every entity – vertex, edge or face – present. One solution is to set up the constraints between the entities, and resolve the dimensions based on these constraints computationally. Many of the constraints can be established automatically, but there may be some that need to be given by the user. Clearly, it is desirable to minimize the user input on such a potentially tedious and error-prone task.

Langbein [Langbein 2004] presented a method in which potential geometric constraints, such as parallelism, planarity and orthogonality, present in a 3D

reverse-engineered model are found approximately at first, followed by a graph-based method to detect inconsistencies between these constraints. The result is a consistent subset of the potential constraints on the initial model. This subset, when resolved numerically, leads to an improvement in the geometry of the model. Langbein's study shows that the graph-based method works well and quickly in detecting inconsistencies.

One fast and simple step in the beautification of an object recovered from a sketch is to enforce planarity on its faces. Chen [Chen 2002] solves this problem using a least-square method to adjust vertices of a face on to the same plane. But he did not take into account the effects of other constraints, such as faces being parallel.

Varley [Varley 2002a] beautified the 3D model reconstructed from a line drawing by determining the face normals and distances between faces separately using an optimization method.

Wilczkowiak *et. al.* incorporated camera calibration and 3D reconstruction of 3D models built from images in computer vision [Wilczkowiak 2003]. The constraints, stored in a graph, are used to find a smaller set of input parameters and functions that yield all the parameters which are used in a subsequent refinement step. The approach is based on a dictionary of rules, called r-methods, that relate geometric entities through constraints, represented as a graph. Attempts are made iteratively to match the graph of an r-method to a subgraph of the constraint graph of the object. Then a reverse sequence of the r-methods identified was executed to solve the constraints. Implicit constraints were not considered because the system cannot contain

redundancies.

The approaches described above beautify a 3D model, but the improved model carries no proper dimensions. Martínez and Félez presented a constraint-based solver to provide a completely dimensioned 2D part [Martínez 2005]. Their method establishes the constraints from two or three sketches of different views of a model, and chooses a set of independent constraints of the system by determining if the system is over-constrained. The constraints form a system of equations, the Jacobian of which, when solved, reveal the redundant constraints. The work does not deal with dimensioning a 3D model.

The user's intent is interpreted and expressed as constraints in the beautification or dimensioning process.

There are limitations in redundant constraint detection using graph-based methods. Graphs have been used to detect structurally redundant constraints without solving the constraint system [Latham 1996; Hoffmann 1998; Li 2002; Langbein 2004; Zou 2005]. However, some constraints are implicit; they can be inferred from a set of constraints in a graph, but are not explicitly represented. Graph-based methods therefore cannot be used to detect all redundant constraints. We demonstrate it using two examples.

Figure 6.2: Pappas's Theorem.

Figure 6.2 illustrates Pappas's Theorem in 2D: If A, B, and C are three points on one line, D, E, and F are three points on another line, and AE meets BD at X, AF meets CD at Y, and BF meets CE at Z, then the three points X, Y, and Z are collinear [Pappas 1989]. The collinear constraint can be inferred from the given set of constraints. The constraint graph of the theorem, which includes all the points, lines, intersections and collinear constraints, is an under-constrained system, and there are no over-rigid subgraphs or redundant constraints given the graph, though in fact the collinear constraint is redundant.

Below is another example in 3D: Let A, B and C be three faces of a block. B and C are perpendicular to A, and if Line L is the intersection of B and C, then L is also perpendicular to A.

Figure 6.3: An example of an implicit constraint in 3D.

L being perpendicular to A can be inferred from it being the intersection of B and C, and therefore would be redundant in a constraint system that includes both.

In a constraint system, many constraints can be inferred from the existing ones.

A constraint system can be expressed as a system of equations [Light 1982], which can be solved numerically. This solution accounts for a large proportion of the cost in geometric constraint solving, which is directly proportional to the size of the system. To minimize this cost, the constraint system is decomposed into smaller sub-systems [Hoffmann 2001a], which are solved separately. The decomposition of constraints requires analyzing the relationships between the weights of entities and weights of constraints, without considering the properties of the entities and constraints [Latham 1996]; implicit constraints are not considered. In the sequence of the partially ordered solvable subgraphs [Hoffmann 2001a], a subgraph may include new constraints which can be inferred from previously solved subgraphs in an under-constrained system. New inferred constraints affect the solution to a subgraph, and hence the solution to the whole system. Therefore, before knowing if a system is well constrained, we cannot ignore the implicit constraints in redundancy detection or decomposition of constraints. When and which implicit constraints are selected for the decomposition is yet another problem.

It is necessary to obtain the inferred constraints in a constraint system. Redundant constraint detection requires the relationships between entities in the known constraint sets. Theorem proving [Buchberger 1985; Chou 1987; Buchberger 2001] can be used to answer “true” or “false” to a specific geometry statement such as

Pappas's Theorem. But it is not possible to use Gröbner basis techniques or Wu's method in redundant constraint detection because both have exponential time complexity. The extension of their methods to 3D cases is possible but expensive.

Because graph-based methods can only detect redundancy from constraints explicit in a graph, a new method is needed for the detection in a graph with implicit constraints.

An increasing amount of literature is devoted to detecting redundancy by numerical methods. Li [Li 2001] and Langbein [Langbein 2002] determined the numerical redundancies of a 3D geometric constraint system by a disturbance method. If the system is unsolvable after adding the disturbance value to a constraint, then the constraint is redundant or inconsistent. If redundancy exists, then the Jacobian matrix of the system of equations is singular [Light 1982]. In the presence of redundancy, that is, when there is a set of mutually dependent constraints, Martínez [Martínez 2005] breaks the redundancy by removing a constraint from the set of dependent constraints. The methods of Light and Martínez work well on 2D cases, but extension to 3D cases is still an open issue.

The main objective of this chapter is the “beautifying” of 3D models reconstructed from 2D sketches, to get the models close to the shapes the designers intend, and give the models proper dimensions. We first introduce the types and expressions for the geometric constraints and the method for finding them, followed by the procedure for selecting a consistent constraint subset from those found. Finally, this constraint subset is used to beautify and dimension the recovered model. Some

experimental results are presented.

6.2 Constraint system

A geometric constraint system can be represented by a set of entities and a set of constraints associated with the entities. The entities contain the variables and the constraints specify a system that allows the variables to be determined.

6.2.1 Entities

The entities are faces, edges and vertices.

A vertex is represented by a three-component vector $\mathbf{p}(x, y, z)$. (We use bold to represent vectors and italics for scalars.)

An edge has four degrees of freedom (two translations and two rotations), represented by six variables and two equations,

$$a^2 + b^2 + c^2 = 1$$

and $ax + by + cz = 0$,

where point $\mathbf{p}(x, y, z)$ is chosen to be the point on the edge nearest to the origin and $\mathbf{d}(a, b, c)$ is the unit vector of the edge direction.

A face has 3 degrees of freedom (one translation and two rotations), represented by a system with six variables and three equations. The six variables are accounted for in the unit face normal $\mathbf{d}(a, b, c)$ and the location $\mathbf{p}(x, y, z)$ which is the nearest point on the face to the origin. $\mathbf{d}$ and $\mathbf{p}$ must be parallel vectors, which means $\mathbf{d} \times \mathbf{p} = \mathbf{0}$. This gives rise to three equations $ay - bx = 0$, $az - cx = 0$ and $bz - cy = 0$, only two of which, say the first two, are independent. The unit face normal requires that

$$a^2 + b^2 + c^2 = 1,$$

thus forming a system of three independent equations. Table 6.1 shows the equations to define the entity and its parameters.

Entity	Parameters	Equations to define the entity
Vertex	location $\mathbf{p}(x, y, z)$	
Edge	edge direction $\mathbf{d}(a, b, c)$ edge position $\mathbf{p}(x, y, z)$, the point on the edge nearest to the origin	$a^2 + b^2 + c^2 = 1$ $ax + by + cz = 0$
Face	face normal $\mathbf{d}(a, b, c)$ face location $\mathbf{p}(x, y, z)$, the point on the face nearest to the origin	$ay - bx = 0$ $az - cx = 0$ $a^2 + b^2 + c^2 = 1$

Table 6.1: Representation of entities.

6.2.2 Constraint equations

The constraints include topological constraints which define the connectivities between entities, and geometric constraints such as distance or angle between entities in a model. Our system specifies only two topological constraints: “vertex on edge” and “vertex on face” directly. Other constraints, such as line on face and face adjacency, are not specified because they can be inferred. For instance, two end points of an edge lying on a face are sufficient to ensure that the edge lies on the face.

A geometric constraint is represented by two positions and one or two directions as shown in Table 6.2, in which $\mathbf{p}_1$ is the point on Entity 1 closest to the origin and $\mathbf{d}_1$ is a unit direction vector, which is the normal for a plane and edge direction for an edge; $\mathbf{p}_2$ and $\mathbf{d}_2$ are the corresponding values for Entity 2. The weight of a constraint is the number of degrees of freedom (DOF) eliminated by the constraint, which is the number of equations required to define that constraint. For example, the constraint of

Geometric constraint		Constraint equations	Weight
Parallel	edge – edge ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 = \pm \mathbf{d}_2$	2
	face – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 = \pm \mathbf{d}_2$	2
	edge – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$	1
Perpendicular	edge – edge ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$	1
	face – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$	1
	edge – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 = \pm \mathbf{d}_2$	2
Distance (h)	vertex – vertex ($\mathbf{p}_1 - \mathbf{p}_2$)	$ \mathbf{p}_1 - \mathbf{p}_2 = h$	1
	Vertex – edge ($\mathbf{p}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$ (\mathbf{p}_1 - \mathbf{p}_2) - ((\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2) \mathbf{d}_2 = h$	1
	Vertex – plane ($\mathbf{p}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = \pm h$	1
	Edge – edge ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 = \pm \mathbf{d}_2$ $ \mathbf{p}_1 - \mathbf{p}_2 = h$	3
	Edge – plane ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$ $(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = \pm h$	2
	Plane – plane ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 = \pm \mathbf{d}_2$ $(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = \pm h$	3
Angle (α)	edge – edge ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = \pm \cos(\alpha)$	1
	face – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = \pm \cos(\alpha)$	1
	edge – face ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = \pm \sin(\alpha)$	1
Position (required by topology of model)	Vertex lie on plane ($\mathbf{p}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = 0$	1
	Vertex lie on edge ($\mathbf{p}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$ (\mathbf{p}_1 - \mathbf{p}_2) - ((\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2) \mathbf{d}_2 = 0$	2
	edge lie on plane ($\mathbf{p}_1, \mathbf{d}_1 - \mathbf{p}_2, \mathbf{d}_2$)	$\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$ $(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = 0$	2

Table 6.2: Constraint equations. $\mathbf{P}_1$ is the point on Entity 1 closest to the origin and $\mathbf{d}_1$ is a direction vector of Entity 1. $\mathbf{P}_2$ and $\mathbf{d}_2$ are the corresponding values for Entity 2.

distance between two vertices requires one equation that eliminates one degree of freedom, thus its weight is 1.

The sign $\pm$ in the table indicates that a value can be either positive or negative, and the choice depends on the initial values of the parameters. For example, the distance between a vertex $\mathbf{p}_1$ and a face $(\mathbf{p}_2, \mathbf{d}_2)$ is $|(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2|$, so the equation describing the constraint is $(\mathbf{p}_1 - \mathbf{p}_2) \cdot \mathbf{d}_2 = \pm h$, where h is a known value. The sign of this value is chosen depending on which half plane of the face the vertex is in. Table 6.1 shows the basic constraints between two entities. Other higher level constraints, such as those listed in Section 2.2, can be expressed as combinations of some of the constraints in the table.

6.2.3 Detecting the constraints

Given a 2D sketch and its reconstructed 3D model, we need to automatically detect all the constraints that may be used to define the model. The types of constraint to be detected are those listed in Table 6.2. Some constraints may be introduced by the user too, if required.

We use the hierarchical clustering algorithm presented by Everitt [Everitt 1986] to find groups of entities that have similar properties. For example, for parallel faces, the clustering into groups proceeds by a series of fusions of faces whose normal vectors are nearly parallel. The normal vectors in a group have a median, and the closeness between groups is measured by the difference between the group medians.

Figure 6.4: Base face of a parallel faces set.

After establishing the parallel face groups, the faces in each group are sorted. In the example in Figure 6.4, faces f_1 , f_2 , f_3 and f_4 are found to be parallel to each other. They are sorted in the order f_1 , f_4 , f_3 and f_2 , according to their position. Then one of the end faces, f_1 or f_2 , is set to be the base face for dimensioning, and only the parallel constraints between the faces in the group and the base face are added to the list of constraints for the object.

Rules such as “two faces are perpendicular if their normals are almost perpendicular” are defined for perpendicular constraints. The constraints of distance and angle are calculated by the equations listed in Table 6.2.

Entity 1	Entity 2	Constraint type	Priority
vertex	vertex	distance between two vertices	1
vertex	edge	distance between vertex and edge	1
		vertex lies on edge	5
vertex	face	distance between vertex and face	2
		vertex lies on face	5
edge	edge	angle between two intersecting edges	1
		parallel & distance between two edges	2
edge	face	parallel & distance between edge and face	3
		angle between edge and face	1
		edge and face perpendicular	2
face	face	angle between two faces	3
		two perpendicular faces	4
		parallel & distance between two faces	4

Table 6.3: Priority values of constraints, a higher value denotes a higher priority.

Choosing the correct type of constraint is essential to establishing the shape [Martínez 2005]. Certain types of constraint are more likely to occur than others. Hence we establish the priority of a constraint according to its type. Table 6.3 shows the priority values defined for the types in our system. A higher value denotes a higher priority. The choice of the values here is partly based on the values given by Langbein [Langbein 2004] and Martínez [Martínez 2005] and partly through our own experience. The priority defined in this Section will be used in selecting the constraints later. Constraints with higher priority will be selected first.

6.3 Selecting and solving the constraints

In commercial CAD systems, the dimensions, which are constraints, of a 3D

model need to be given by the user. It is up to the user to provide a complete set of dimensions.

In our system, a large number of constraints are detected using the method described in Section 6.2.3. The user may also choose to introduce a constraint directly. Only a subset of these constraints is needed to define the model completely. The result should be a well-constrained system containing all the entities and the selected constraints.

Topological constraints - “vertex on face” and “vertex on edge” - are a part of the structure of a model and can be identified correctly and directly from the model.

Geometric constraints are selected subsequently, with constraints with higher priority selected first. A new constraint is accepted if it is not redundant or inconsistent with the existing ones. Otherwise, this new constraint or one of the existing ones is eliminated. Redundancy may be structural or numerical. This process repeats until the difference between the sum of the weights of the entities and the sum of the weights of the constraints is six, because a 3D model has six degrees of freedom, three rotations and three translations. If the last added constraints are redundant or inconsistent with the existing constraints, then the last constraint or one of the existing constraints will be deleted to make the whole system not over-constrained. So, it is impossible that the system will end up with any part of the model over-constrained. The final system will be a well-constrained model with six degrees of freedom. Finally, the constraint equations are solved and the new model is obtained by updating the parameters of the entities.

6.3.1 Solving the constraints

Given a set of entities and constraints, we want to derive the values of the parameters that define the entities, which satisfy all the constraints. When a system is well-constrained or even under-constrained, the parameters can be found. This section focuses on solving the constraints through numerical optimization.

Let $f_k(\mathbf{x}) = 0$ ($k = 1 \dots m$) be a set of possibly non-linear equations defining the constraints described earlier, where the variable $\mathbf{x}$ is a vector of the parameters defining the entities of the model (see Table 6.1). Typically, $\mathbf{x}$ contains many elements, the number of which increases with the complexity of the model. This is therefore a multivariate optimization problem. Given the m equations, a solution for $\mathbf{x}$ can be found using a least-square method, by minimizing the objective function:

$$\sum_{k=1}^m (f_k(\mathbf{x}))^2 .$$

The initial value of $\mathbf{x}$ for the optimization can be obtained from the reconstructed model; the solution at each step of the optimization forms the input to the next.

When the objective function value approaches zero, the optimization approaches a configuration of the model that satisfies all the constraints. Failure to arrive at zero means that the constraint system is not solvable, which means there are conflicting constraints in the system.

Some global optimization techniques, such as genetic algorithm, simulated annealing, branch and bound [Gray 1997], are effective even when the objective function is very complex, but they are highly time-consuming, and therefore are not suitable for an interactive real-time application, which we intend our system to be.

The BFGS method [Deb 1995], which is mature, stable and effective, has been used in this project. Derived from the method of steepest descent, BFGS is a Quasi-Newton method based on the idea of reconstructing a quadratic approximation of a function from values of its gradients at a number of points, leading to a better approximation of the minimum value. The BFGS method cannot handle multiple discrete solutions. But the initial input of the optimization method described in Chapter 6 may affect which solution the method will return for a problem with multiple discrete solutions. The reconstructed 3D model is the input for the beautification process. The constraints used in reconstruction include some global ones such as MSDA, which are helpful to selecting the right one from the multiple discrete solutions.

Next Section will turn to the details of checking if the last constraint selected is redundant or inconsistent.

6.3.2 Eliminating structurally redundant constraints

As has already been mentioned, it is difficult for graph-based methods to detect all the numerically redundant constraints, but it can detect the structurally redundant ones without solving the constraint equations. If a constraint is structurally redundant, then it must be numerically redundant also. Hence, it is useful to use a graph-based method described in the previous chapter to detect the structural redundancy. Basically, it uses a flow-based algorithm to identify the existence of structural redundancy and over-constrained subgraphs in a 2D or 3D constraint system. It exploits the degree of freedom and degree of rigidity properties of subgraphs, and identifies the structurally

over-constrained ones by adding the constraints one by one to the entities in the graph.

6.3.3 Detecting numerical redundancy

A set of constraints can be expressed as a set of equations, $f(\mathbf{x}) = 0$, where $\mathbf{x} = (x_1, \dots, x_n)$ is a vector of n variables. More explicitly, the equations can be written as

$$\begin{cases} f_1(x_1, \Lambda, x_n) = 0 \\ \quad \quad \quad \mathbf{M} \\ f_m(x_1, \Lambda, x_n) = 0 \end{cases}$$

where m is the number of constraints. The Jacobian matrix is then

$$J(x_1, \Lambda, x_n) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \Lambda & \frac{\partial f_1}{\partial x_n} \\ \mathbf{M} & \mathbf{O} & \mathbf{M} \\ \frac{\partial f_m}{\partial x_1} & \Lambda & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}} \\ \mathbf{M} \\ \frac{\partial f_m}{\partial \mathbf{x}} \end{bmatrix}.$$

If a constraint, with equation $g_k(\mathbf{x}) = 0$, is redundant, then it can be deduced from a subset of $f(\mathbf{x}) = 0$, $g_1(\mathbf{x}) = 0$, ..., $g_l(\mathbf{x}) = 0$, say. We can express $g_k(\mathbf{x})$ as a function of $g_1(\mathbf{x})$, ..., $g_l(\mathbf{x})$:

$$g_k(\mathbf{x}) = F(g_1(\mathbf{x}), \dots, g_l(\mathbf{x})). \quad (6.1)$$

When $g_1(\mathbf{x}) = 0$, ..., $g_l(\mathbf{x}) = 0$, then $g_k(\mathbf{x}) = 0$.

By differentiating both sides of Equation 6.1, we obtain

$$\frac{\partial g_k}{\partial \mathbf{x}} = \frac{\partial F}{\partial g_1} \frac{\partial g_1}{\partial \mathbf{x}} + \frac{\partial F}{\partial g_2} \frac{\partial g_2}{\partial \mathbf{x}} + \Lambda + \frac{\partial F}{\partial g_l} \frac{\partial g_l}{\partial \mathbf{x}}. \quad (6.2)$$

When $f(\mathbf{x}) = 0$, that is, $g_1(\mathbf{x}) = 0$, ..., $g_l(\mathbf{x}) = 0$, $\frac{\partial F}{\partial g_1}, \frac{\partial F}{\partial g_2}, \dots, \frac{\partial F}{\partial g_l}$ are constants.

To simplify the notation, we denote $\frac{\partial F}{\partial g_1}$ by c_1 , $\frac{\partial F}{\partial g_2}$ by c_2 , ..., $\frac{\partial F}{\partial g_l}$ by c_l .

So Equation 6.2 becomes

$$\frac{\partial g_k}{\partial \mathbf{x}} = c_1 \frac{\partial g_1}{\partial \mathbf{x}} + c_2 \frac{\partial g_2}{\partial \mathbf{x}} + \Lambda + c_l \frac{\partial g_l}{\partial \mathbf{x}} \quad (6.3)$$

Thus, if an equation of a constraint $g_k(\mathbf{x}) = 0$ in the system is numerically redundant, its corresponding row vector in the Jacobian matrix can be expressed as a linear combination of the other row vectors, and the Jacobian matrix is rank deficient as well.

The constraints with the highest priority are added to the system one by one. So the last row of the Jacobian matrix corresponds to the newly added constraint. The condition for a non-redundant constraint is that the Jacobian is not rank deficient. Let $f_m(\mathbf{x}) = 0$ be the last constraint added and $J_1, J_2 \dots J_m$ be the row vectors of the Jacobian matrix. If the last constraint is redundant, then $J_1, J_2 \dots J_{m-1}$ of the Jacobian matrix are linearly independent and $J_1, J_2 \dots J_m$ are linearly dependent. So that we can write

$$J_m = c_1 J_1 + c_2 J_2 + \dots + c_{m-1} J_{m-1}.$$

That is, $[J_1^T, J_2^T \dots J_{m-1}^T][c_1, c_2 \dots c_{m-1}, -1]^T = 0$, where $c_1, c_2 \dots c_{m-1}$ are scalars.

We detect the redundancy by solving the equation

$$J^T \mathbf{c} = 0, \quad (6.4)$$

where $\mathbf{c} = [c_1, c_2 \dots c_{m-1}, -1]^T$ is a vector of the unknowns and J^T is a $n \times m$ matrix. If there is no $\mathbf{c}$ for which Equation 6.4 holds, then the m row vectors are linearly independent. Otherwise, they are linearly dependent, because then we can express the last row vector of J as a linear combination of the other row vectors.

Equation 6.4 needs to be solved, in which J^T is n by m , with $n \geq m$ always. There

are more equations than unknowns, so the system is over-determined. We will briefly review the techniques related to solving sparse over-determined systems below.

A direct method may often be preferable to an interactive method for solving an over-determined problem [Paige 1982]. Normally, methods based on normal equations and QR factorization are used to solve an over-determined problem [Golub 1996]. Golub's investigation into solving over-determined systems showed that the flop (floating point operation) of the method via normal equations is $mn^2 + n^3/3$ and that via Householder orthogonalization is $2mn^2 - 2n^3/3$ when the matrix in the system is $m \times n$. For a full rank system, it is difficult to choose the right algorithm from the two. Note also that J^T may be rank deficient when the last constraint is redundant, which raises another issue about selecting a numerical rank determination technique. The QR factorization method can be used to determine the rank of J^T ; the method based on normal equations will fail in rank deficient cases.

In the system, J^T may be rank deficient. There are a number of ways to compute the rank of a matrix. The singular value decomposition algorithm is very time-consuming. Bischof developed an algorithm for rank revealing orthogonal factorization (RRQR). His experimental results showed his approach performs up to three times faster than the less reliable QR factorization with column pivoting used in LAPACK [Bischof 1998], a package popular for computing rank-revealing factorizations. Their approach comes within 15% of the speed performance of the LAPACK for computing a QR factorization without any column exchanges, which is implemented in the DGEQRF routine of LAPACK.

The Jacobian matrix is a sparse matrix with about 9 non-zeros in each row on average. All the constraint equations have less than 9 variables (see Table 6.2), so most elements in a row vector of the Jacobian matrix are zero and only elements corresponding to the variables are non-zero; that is, about 9 non-zeros in each row. The exact cost of solving the equations depends strongly on the structure of the problem, and efficient computation of rank-revealing decompositions of sparse matrices is an open area of research. Berry [Berry 2005] computed sparse pivoted QR approximation by the Quasi-Gram-Schmidt algorithm to obtain low-rank approximations. His method is slow when $\text{rank} = \min(m, n) - 1$, needing about 14s to obtain the rank of a 500×400 matrix. Methods for sparse QR decomposition are also an area of active research. For sparse QR decomposition, Matlab [Mathworks 2006] uses a C program written by Gilbert [Gilbert 1992]. From our experiments using Matlab, when A is a sparse 500×400 matrix with 4500 non-zero elements, the run time for QR decomposition is almost the same as that of QR decomposition of a dense matrix. When the size of A is smaller, QR decomposition for a dense matrix is much faster than sparse QR decomposition.

From this perspective, the solution for the linear system will be computed using QR factorization for dense matrices in our system. The idea of factoring $J^T = QR$ is used, where Q is orthogonal and R is the upper triangular, and Q is n by n and R is n by m . Then $J^T \mathbf{c} = QR\mathbf{c} = 0$ is solved by

$$R\mathbf{c} = 0, \tag{6.5}$$

where $\mathbf{c} = [c_1, c_2, \dots, c_{m-1}, -1]^T$, is a vector of the unknowns.

In contrast to general rank detection of a matrix, we know in advance that the last equation introduces the redundancy. So we detect the redundancy by simply using R_{mm} , which is the entry in the m th row and m th column of the upper triangular matrix R . When the magnitude of R_{mm} is not zero, then it is impossible for Equation 6.4 to have a solution, because the m th equation of the set in Equation 6.5 is $R_{mm}*(-1) = 0$.

If R_{mm} is not zero, then Equations 6.4 and 6.5 have no solutions; that means the last line vector of the Jacobian matrix is not a linear combination of the other line vectors and the last constraint is not redundant. When rank deficiency exists, we can detect it from the R of the QR factorization, without solving the equations. That is another reason for choosing QR factorization.

In our method, the redundancy or rank of the matrix J is determined by QR factorization of J only, without using another special technique for rank revealing such as SVD or QR with column pivoting. In our system, Householder transformations are utilized for QR factorization.

The flow chart for determining if the last constraint is redundant or inconsistent is shown in Figure 6.5.

Figure 6.5: Flow chart for determining if a constraint is redundant or inconsistent.

If the last constraint is found to be structurally redundant by the graph-based method, it is rejected directly without invoking the numerical method described above. Otherwise, the action moves to the numerical stage. When a new constraint is added to the system, we need to solve the system of constraint equations to obtain the values of the entries in the Jacobian matrix. Because the cost for QR factorization of the Jacobian matrix is small compared to that for solving the equations, we try to avoid solving the equations. But this is possible only when the last constraint is distance or

angle between entities, because the values of such constraints can be obtained from the existing entities in the model.

6.3.4 A simple example for redundancy detection

A simple example is given below to demonstrate the general working principles in redundancy detection. The example object has 3 faces, 9 edges and 7 vertices (Figure 6.6). Using Table 6.1, we see that these entities contain $3 \times 6 + 9 \times 6 + 7 \times 3 = 93$ variables and $9 \times 2 + 3 \times 3 = 27$ constraints within themselves. There are $4 \times 3 = 12$ “vertex on face” and $2 \times 9 = 18$ “vertex on edge” topological constraints; the weight of the former is 1 and that of the latter is 2 (see Table 6.2), thus giving a total of $12 + 2 \times 18 = 48$ topological constraints, which are identified automatically directly from the structure of the graph. Hence there are $27 + 48 = 75$ constraints in the structure. We demonstrate redundancy detection by adding three other constraints one by one (thus giving 78 constraints in all). Two of these constraints, Face f_0 perpendicular to Face f_1 , f_0 perpendicular to Face f_2 , are introduced first. There are no structural or numerical redundancies, so the two constraints are accepted.

Figure 6.6: An example with numerical redundant constraint.

The final constraint is Edge e_6 perpendicular to f_0 . It is not structurally redundant; we thus focus on numerical redundancy. The variables and the constraints result in a 78×93 Jacobian matrix. Table 6.4 illustrates its structure, in which asterisks denote components which are not zero. To simplify the table, we use an entity name such as v_0 to represent all the parameters of each entity, instead of the parameters themselves. Each row indicates the row vector for the equation of a constraint. The inherent constraints of an edge or a face are the equations defining the entity given in Table 6.1, and their corresponding row vectors are in the top rows of Table 6.4. These are followed by the row vectors of the equations for the topological constraints. The last three rows are the vectors for the equations of the two vertical faces and the vertical edge between the two faces.

After the QR factorization of the Jacobian matrix, the element $R_{78,78}$, which is the entry in the 78th row and 78th column of the upper triangular matrix R , is zero. That means the last constraint is numerically redundant.

		v0	v1	v2	v3	v4	v5	v6	e0	e1	e2	e3	e4	e5	e6	e7	e8	f0	f1	f2
Constraints in edge (18)	Extra e_0								*											
	...																			
	Extra e_8																	*		
Constraints in face (9)	Extra f_0																	*		
	Extra f_1																		*	
	Extra f_2																			*
vertices on face (12)	v_3 lie on f_0				*													*		
	...																			
	v_1 lie on f_2		*																	*
Vertices on line (36)	v_0 lie on e_0	*							*											
	...																			
	v_5 lie on e_8						*										*			
Constraints detected (3)	$f_0 \perp f_1$																	*	*	
	$f_0 \perp f_2$																	*		*
	$l_6 \perp f_0$														*			*		

Table 6.4: The Jacobian Matrix for the example of Figure 6.6. * denote non-zeros elements.

By solving Equation 6.5, the vector of the linear combination can be obtained:

[illegible]

Not all the elements in $\mathbf{c}$ are zero. In the Jacobian matrix, the row vector of the last constraint is a linear combination of the row vectors which correspond to the non-zero elements in $\mathbf{c}$. That is, if the i th element in $\mathbf{c}$ is non-zero, then the i th row vector in the Jacobian matrix is a member of the linear combination upon which the last row vector is dependent. Hence, in the example, the linear constraint of $\mathbf{e}_6$ perpendicular to $\mathbf{f}_0$, which is the last constraint added, is dependent on these eight constraints: $\mathbf{v}_5$ on $\mathbf{f}_1$, $\mathbf{v}_2$ on $\mathbf{f}_1$, $\mathbf{v}_5$ on $\mathbf{f}_2$, $\mathbf{v}_2$ on $\mathbf{f}_2$, $\mathbf{v}_5$ on $\mathbf{e}_6$, $\mathbf{v}_2$ on $\mathbf{e}_6$, $\mathbf{f}_0$ perpendicular to $\mathbf{f}_1$ and $\mathbf{f}_0$ perpendicular to $\mathbf{f}_2$. The system or user can select and delete one constraint from the group, except the topological constraints.

The cost of every procedure for checking for redundancy in the example is shown in Table 6.5. This result shows that the cost in solving the constraints is the largest among all the procedures. It is the same in other more complex examples. One observation we can make is that the system of equations is large. Given the very simple example here, the Jacobian matrix is already 78×93 in size. Hence an efficient solution is very important.

Last constraint added	Checking structural redundancy	Solving the constraints	Checking numerical redundancy by QR factorization of the Jacobian matrix	Total
$f_0 \perp f_1$	0.015	0.375	0.031	0.421
$f_0 \perp f_2$	0.016	0.312	0.031	0.359
$e_6 \perp f_0$	0.015	0.265	0.031	0.311

Table 6.5: Time (in seconds) for redundancy detection

6.4 Changing dimensions by the user

After setting up the constraints through the procedures described above, the object is well-constrained. At this point, the dimensions in the object are directly derived from the drawing, which do not normally correspond to the dimensions the designer has in mind.

The user can establish a relationship between the size of the reconstructed model and the size that the user requires by specifying the correct value to a known dimension. This then provides a scaling factor, which can be applied easily to scale the whole object to the required size. After that, the user can manually reset the value of any individual dimension if so required. Finally, the constraint system can be solved again to generate a rectified model.

The running time for constraint solving of a well-constrained system can be reduced by decomposing the constraint problem into smaller ones. There are many methods that can be used to decompose the constraints of a well-constrained system [Latham 1996; Hoffmann 2001b; Sitharam 2004; Gao 2006]. The method of Latham and Gao is employed in our system. By analyzing the connectivities between constraints and the geometric entities which they constrain, construction sequences can be generated. After the decomposition, the time needed to solve the constraint equations decreases greatly. So, when the user changes some values of dimensions, the system reacts much faster.

6.5 Examples

The examples in Figure 6.7 show some results from our system that implements our method of beautification and dimensioning of a reconstructed 3D model. The system was implemented in Visual C++ with OpenGL, running on a 2.8G Pentium 4 PC.

There are too many constraints detected for each example, so we do not present them in detail. In the figures below, the original reconstructed 3D model, the model after beautification and the dimension are listed. In the examples, all the vertices of a model lie on faces, all the constraints selected are satisfied. Table 6.6 lists the time cost for selecting the constraints for beautification and dimensioning, the time for solving an equation set, the time for solving an equation set after constraint decomposition, the time for determining structural redundancy and then for numerical

redundancy when a new constraint is added. It can be seen from the table that when the model is more complex, the time for detecting a correct set of constraints is greater, as expected. The constraint solving process is the most time-consuming. Fortunately, after the constraint decomposition, the processing time decreases dramatically. That is fast enough for interactive design.

Object	Number of faces	Number of constraints*	Selection time	Solving time	Solving time after constraint decomposition	Checking structural redundancy of a new constraint	Checking numerical redundancy of a new constraint
1	6	6	4.57	0.67	0.09	0.01	0.06
2	8	8	12.90	1.32	0.11	0.01	0.17
3	8	10	24.18	1.96	0.60	0.03	0.17
4	9	9	19.93	2.56	0.14	0.03	0.25
5	10	10	30.00	2.96	0.20	0.03	0.35
6	7	9	8.51	1.09	0.23	0.01	0.10
7	8	10	18.90	1.65	0.37	0.01	0.17
8	10	9	35.82	3.15	0.18	0.03	0.37
9	14	16	122.89	12.84	0.23	0.09	1.17
10	14	16	56.42	9.81	0.25	0.09	1.18
11	9	13	34.35	2.39	1.11	0.03	0.26
12	14	14	52.61	9.79	0.15	0.09	1.15
13	10	12	40.78	3.43	0.56	0.03	0.37
14	10	10	23.36	2.81	0.20	0.04	0.36
15	8	12	29.64	2.64	0.17	0.03	0.17
16	10	10	25.73	1.57	0.15	0.03	0.17

Table 6.6: Computing times (seconds) for objects in Figure 6.7 (* Exclude topological constraints).

Figure 6.7: Test examples. The left line drawing for each object is the input to our beautification system, and the centre drawing is the output. The right shaded figure is not generated by our system but created separately to show the dimensions (or constraints) that have been found and retained, as we have yet to include the posting of dimensions in our system.

6.6 Symmetry and beautification

Skewed symmetry is a useful constraint in 3D reconstruction and the subsequent beautification of the 3D model. We have studied skewed symmetry detection in Chapters 3 and 4.

The incorporation of symmetry would result in a different set of constraint equations. Symmetry would be a constraint of a high priority and would result in a reduction of the problem size, as only $1/n$ of the object needs to be constrained and solved for an n -fold rotationally symmetric object.

Symmetry has not been incorporated as a constraint for beautification in this thesis. This is not due to the lack of intention or foresight, but to the limited time span in a PhD candidature. Clearly, symmetry is a very important constraint and should be incorporated; the dimensioning of some of the examples in Section 6.5 would have turned out differently if it had been.

6.7 Conclusions

This chapter establishes an algorithmic framework for beautification and automatic dimensioning of a 3D model recovered from a 2D sketch. The recovered model may be “rough”, given that a sketch is often inaccurate. The experiments show that our method beautifies the recovered model very well, to something more akin to a well-designed object. It also generates the dimensions correctly. The dimensions can be modified if they do not meet the design requirements.

The method for detecting redundant constraints introduced in this chapter still

applies to a symmetric object, except that it is applied to a smaller, and therefore computationally simpler problem.

CHAPTER 7

Summary and Future Work

This chapter summarizes the results of the research and the contributions to the field. This is followed by recommendations for future work.

7.1 Summary of accomplishments

The work in this thesis contributes to the effort of reconstructing 3D models from 2D sketches of planar polyhedral objects, the goal of which is to provide a more natural user interface to support conceptual design through sketching in a software system. There are two main contributions: the detection of skewed mirror and rotational symmetry in a sketch and a method to beautify the reconstructed model.

7.1.1 Skewed symmetry detection

Symmetry is an important constraint in the 3D reconstruction of an object from a 2D sketch and the subsequent beautification of the recovered 3D model. There are two types of symmetry: mirror and rotational. Skewed symmetry is symmetry that exists in a projection. In this thesis we provide new algorithms to detect both skewed mirror and rotational symmetries in 2D sketches. The drawing, which includes all the edges of the object, visible and hidden, is first converted into an edge-vertex graph. From the graph, our algorithm first finds the faces of the object, and then establishes that the faces, edges and vertices are topologically symmetric. True symmetry is then established by checking that the topologically symmetric vertices are also

geometrically symmetric. While the algorithms for detecting topologically symmetric sets are similar for both mirror and rotational symmetry, their algorithms for geometric symmetry are different.

For mirror symmetry, geometric symmetry is established by making sure that all the lines joining the topologically symmetric vertices across the mirror plane are parallel. In the case of rotational symmetry, vertices that are topologically symmetric must be evenly distributed on an ellipse in the sketching plane, and in the case of multiple sets of topologically symmetric vertices, their ellipses must have collinear centres, parallel axes and the same ratio between their major and minor axes. A tolerance is given in the calculation of these geometric relationships, to allow for inaccuracies in the line drawings. The algorithms have been implemented and the experimental results show that our method is stable and effective in symmetry detection.

7.1.2 Beautification and dimensioning of a recovered model

3D models reconstructed from 2D sketches are inaccurate because of the inherent inaccuracies in the input and the reconstruction methods. It is therefore necessary to beautify them before they can be used in CAD systems. The algorithm presented in this thesis for beautification detects geometric constraints present in the reconstructed model, such as parallel and orthogonal faces, and extracts from these constraints a consistent and sufficient subset that completely defines the object. This subset is arrived at by systematically detecting and removing redundant and inconsistent constraints. In 2D, several different approaches have been proposed for redundancy

detection, but little has been reported for 3D. This thesis has presented two complementary techniques for redundancy detection in the constraints systems: a graph-based method and a numerical method. The graph-based method checks for structural redundancy and decomposes the graph if possible. This is followed by checking for numerical redundancy, and each part of the decomposition is examined further with the numerical method. The result is a set of consistent constraints, which defines completely the dimensions of the model. In particular, the method can be used to detect redundant or inconsistent constraints in under-constrained and over-constrained problems. Furthermore, the methods can be extended to a general framework to cover more general geometric constraint problems in the applications. The algorithms have been implemented and the test results show that they perform well.

7.2 Discussions and future work

The work reported in this thesis is only a part of the greater task of 3D reconstruction from 2D sketches. Within the two areas of major contributions, skewed symmetry detection and beautification, further work is necessary. The constraint of time within a PhD candidature means that some useful tasks have to be left undone. They are listed here as future work. Some of these have been mentioned in the earlier chapters, and some are new.

1. Fewer restrictions in the sketches

It has been stated at the outset that the input sketch of a 3D object is assumed to

be in parallel projection and that all edges must be drawn. These are rather strong restrictions which may render the resulting system less useful. It is therefore necessary to investigate the issue of accommodating perspective projection drawings which have hidden lines removed. As has been pointed in Chapter 3, perspective projections can be transformed into parallel projections, but this has yet to be implemented in our system.

Dealing with a drawing without hidden lines is much more difficult, as our algorithm operates only on the data present. Not drawing hidden lines means that some data is not available, and the symmetry detection algorithms would need to be modified to establish symmetry of part of the data present and the possibility of the missing data being symmetric to the remaining part. This is a major research issue, which might require techniques in artificial intelligence and computer vision.

2. Inferring rotational symmetry from mirror symmetries

Relationships exist between mirror symmetry and rotational symmetry. We can infer rotational symmetry from mirror symmetries in some cases. However not all rotational symmetries can be inferred from mirror symmetries because some models are rotationally symmetric, but not mirror symmetric and vice versa. Figure 7.1(a) gives an example, in which the axis of the 2-fold rotational symmetry is the line of intersection between the symmetry planes of the two mirror symmetries. More work is required to establish clearly the relationship between the two types of symmetries, and using these rules where they are applicable, we can then save on the computation for rotational symmetries.

Figure 7.1 Mirror symmetry and rotational symmetry.

3. Partial symmetry

When a model is partially symmetric only some of the elements in the model are symmetric to each other. Many real-life objects are only partially symmetric and it is therefore useful to be able to detect partial symmetry. Figure 7.2 shows a 2D [Shen 2001] and a 3D example of partial symmetry. The method of detecting skewed symmetry may be applicable to partial symmetry detection by loosening the rules of geometrical symmetry when the model is topologically symmetric.

Figure 7.2: Examples of partial symmetry.

4. A better optimization method

It should be noted that the BFGS algorithm described in Section 6.3.1 may fail if the initial input is too far away from the desired solution. That may happen when the

initial model for the start point of the optimization is very different from the model defined by the constraints. Thus a good initial variable vector is important. In all our experiments, because the sketches are discernable, the initial value vectors extracted from the recovered models are good enough for the beautification. But if the user interferes by drastically changing some dimensions, there is no guarantee that the BFGS algorithm will deliver the right solution. Though the method described in this thesis is more reliable for redundant constraint detection compared to graph-based methods, the running time, especially in constraint solving, needs to be improved still. A new speedy optimization method or a faster computer would help.

Multi-dimensional non-linear optimization is a subject for extensive research. This study may have possible wider relevance to the method of solving nonlinear equations with multiple solutions. The problem merits further investigation.

5. Constraint selection

This thesis has focused attention on detecting redundant/inconsistent constraints and selecting constraints to form a well-constrained system. However, this does not mean that using the simple priority introduced in Table 6.3 will always lead to the correct dimensioning of a model. It should be noted that the system being well-constrained is a sufficient but not necessary condition to a correct dimension set. The problem of how to define the priority of detected constraints is complex. Much work remains to be done. Defining the priority of the constraints according to the type of model may improve the result of constraint selection, because the priorities for different types of model may be different. Other factors, like standards or rules on 3D

dimensioning, will also affect the constraint selection.

6. Higher level constraints

The constraints discussed in the redundancy detection of this thesis are basic constraints (angle, distance, parallelism) between basic entities (vertex, edge and face). A higher level constraint such as symmetry and feature can be expressed by a combination of such basic constraints. So the numerical method to identify the redundant constraint introduced in Chapter 6 can also be applied to the constraint of skewed mirror and skewed rotational symmetry. Incorporating symmetry in the beautification process is an implementation issue, not a research issue, as noted in Section 6.6. Other high level constraints such as features can be included in the beautification process too. We are considering a solution based on feature identification within the model and applying the constraints within and between features. But higher level constraints may raise problems in detection of structurally redundant constraints introduced in Chapter 5, because currently the method can only be applied to the constraint between two entities only.

7. Locating the dimensions

As has already been noted, once identified, the dimensions need to be represented and posted based on some standard for dimensioning. Commercial CAD systems already have good and effective methods for posting dimensions. It is therefore no longer a research issue. Nonetheless, for completeness, it would be good to implement the same features in our SRLD system too.

8. Symmetry and beautification

As noted in Section 6.6, symmetry serves to reduce the size of the problem in beautification, by restricting the process to only one part in the symmetry. There do not seem to be deep theoretical issues in making the reduction. The main work is in the implementation, which has yet to be done.

9. Curved objects

Curved objects, rather than planar polyhedra, are the class of objects commonly produced by designers. Our current research has not dealt with curved objects, and more research is therefore necessary for this work to reach its ultimate conclusion. It is not obvious that the solutions presented in this thesis on the various problems are directly extendible to curved objects. But it is possible that the method of skewed symmetry detection can be extended to objects with curved faces by changing the rules in detection symmetry face pairs. Multiple points on a curved face may be used to identify symmetric curved faces. In the beautification process, detecting and expressing the constraints related to curved faces needs further investigation. It is possible that the numerically redundant constraint detection method of this thesis can be used directly to objects with curved faces if the constraints related to the curved constraints can be expressed by equations.

7.3 Conclusion

Currently, the reconstructed object does match the depicted object, but the reconstruction process is prone to errors due to the inaccuracies that exist in the sketch.

The reconstructed model can be improved in the beautification process. And our method of identifying and solving the constraints has proved a feasible means of beautifying the reconstructed model. But much work still remains to be done on the subject before we can have a usable end user system.

References

- [Adobe] Adobe. Adobe Photoshop 7.0 On-line Manual. Chapter on transforming perspective.
- [Ait-Aoudia 1993] Ait-Aoudia S., Jegou R. and Michelucci D. Reduction of constraint systems. *Compugraphics*, Alvor, Portugal: 83-92, 1993.
- [Aldefeld 1988] Aldefeld B. Variations of geometries based on a geometric-reasoning method. *Computer-Aided Design*, 20: 117-126, 1988.
- [Anderl 1996] Anderl R. and Mendgen R. Modelling with constraints: theoretical foundation and application. *Computer-Aided Design*, 99: 155-168, 1996.
- [Barrow 1981] Barrow H.G. and Tenenbaum J.M. Interpreting line drawings as three-dimensional surfaces. *Artificial intelligence*, 17: 75-116, 1981.
- [Benko 2002] Benko P., Kos G., Varady T., Andor L. and Martin R. Constrained fitting in reverse engineering. *Computer Aided Geometric Design*, 19: 173-205, 2002.
- [Berry 2005] Berry M.W., Pulatova S.A. and Stewart G.W. Algorithm 844: computing sparse reduced-rank approximations to sparse matrices. *ACM Transactions on Mathematical Software*, 31: 252-269, 2005.
- [Bidarra 1999] Bidarra R. Validity maintenance in semantic feature modeling. PhD thesis. The Netherlands. Delft University of Technology. 1999.
- [Bidarra 2000] Bidarra R. and Bronsvoort W.F. Semantic feature modelling. *Computer-Aided Design*, 32: 201-225, 2000.
- [Bier 1990] Bier E.A. Snap-dragging in three dimensions. In: *Proceedings of the ACM SIGGRAPH, Computer Graphics*, 193-204, 1990.

- [Bischof 1998] Bischof C.H. and Quintana-Ortí G. Computing rank-revealing QR factorizations of dense matrices. *ACM Transactions on Mathematical Software*, 24: 226-253, 1998.
- [Bloomenthal 1998] Bloomenthal M., Zeleznik R.C., Fish R., Holden L.S., Forsberg A.S., Riesenfeld R., Cutts M., Drake S., Fuchs H. and Cohen E. Sketch-N-Make: automated machining of CAD sketches. In: *Proceedings of ASME Design Engineering Technical Conferences*, 1-11, 1998.
- [Bouma 1995] Bouma W., Fudos I., Hoffmann C.M., Cai J. and Paige R. A geometric constraint solver. *Computer-Aided Design*, 27: 487-501, 1995.
- [Brent 1973] Brent R.P. *Algorithms for minimization without derivatives*. Englewood Cliffs, New Jersey: Prentice-Hall, 1973.
- [Bruderlin 1987] Bruderlin B. Rule-based geometric modelling. PhD thesis. Switzerland. 1987.
- [Buchanan 1998] Buchanan S.A. and de Pennington A. Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems. *Computer-Aided Design*, 25: 741-750, 1998.
- [Buchberger 1985] Buchberger B. Grobner bases: an algorithmic method in polynomial Ideal theory. In Bose N.K. (Ed.) *Multidimensional Systems Theory*: D. Reidel Publishing Co, 1985.
- [Buchberger 2001] Buchberger B. Grobner bases: a short introduction for systems theorists. In: *EUROCAST 2001 (8th International Conference on Computer Aided Systems Theory - Formal Methods and Tools for Computer Science)*, Las

Palmas de Gran Canaria, 2001.

[Cham 1995] Cham T.J. and Cipolla R. Symmetry detection through local skewed symmetries. *Image and Vision Computing*, 13: 439-450, 1995.

[Chen 2002] Chen X.C. Planarity correction for object constructed from line drawings. Master thesis. Singapore. Nanyang Technological University. 2002.

[Chou 1987] Chou S.C. Mechanical geometry theorem proving. Norwell, MA: Kluwer Academic Publishers, 1987.

[Chung 2000] Chung J.C.H., Hwang T.-S., Wu C.-T., Jiang Y., Wang J.-Y., Bai Y. and Zou H. Framework for integrated mechanical design automation. *Computer-Aided Design*, 32: 355-365, 2000.

[Clowes 1971] Clowes M.B. On seeing things. *Artificial Intelligence*, 2: 79-112, 1971.

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

[Csabai 1996] Csabai A., Taiber J. and Xirouchakis P. Design support using constraint-driven design spaces. In: *Workshop on Geometric Constraint Solving & Applications*, Technical University of Ilmenau, 76-84, 1996.

[Darga 2004] Darga P.T., Liffiton M.H., Sakallah K.A. and Markov I.L. Exploiting structure in symmetry detection for CNF. In: *Design Automation Conference (DAC)*, San Diego, CA, 530-534, 2004.

[Deb 1995] Deb K. Optimization for engineering design: algorithms and examples: Prentice-Hall of India, 1995.

- [Dohmen 1998] Dohmen M. Constraint-based feature validation. PhD thesis. The Netherlands. Delft University of Technology. 1998.
- [Durand 2000] Durand C. and Hoffmann C.M. A systematic framework for solving geometric constraints analytically. *Journal of Symbolic Computation*, 30: 493-519, 2000.
- [Eggl 1997] Eggl L., Hsu C.-y., Bruderlin B.D. and Elber G. Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29: 101-112, 1997.
- [Everitt 1986] Everitt B. Cluster analysis. London: Heinemann Educational Books Ltd., 1986.
- [Fitzgibbon 1999] Fitzgibbon A.W., Pilu M. and Fisher R.B. Direct least-squares fitting of ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21: 476-480, 1999.
- [Fudos 1998] Fudos I. Constraint solving for computer aided design. PhD thesis. West Lafayette, IN, USA. Purdue University. 1998.
- [Fudos 1996] Fudos I. and Hoffmann C.M. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry & Applications*, 6: 405-420, 1996.
- [Fudos 1997] Fudos I. and Hoffmann C.M. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16: 179-216, 1997.
- [Fukui 1988] Fukui Y. Input method of boundary solid by sketching. *Computer-Aided*

- Design, 20: 434-440, 1988.
- [Gao 1998a] Gao X.S. and Chou S.C. Solving geometric constraint systems. II, a symbolic approach and decision of rc-constructibility. Computer-Aided Design, 30: 115-122, 1998a.
- [Gao 1998b] Gao X.S. and Chou S.C. Solving geometric constraint systems. I. a global propagation approach. Computer-Aided Design, 30: 47-54, 1998b.
- [Gao 2004] Gao X.S., Hoffmann C.M. and Yang W.Q. Solving spatial basic geometric constraint configurations with locus intersection. Computer-Aided Design, 36: 111-122, 2004.
- [Gao 2006] Gao X.S., Lin Q. and Zhang G. A C-tree decomposition algorithm for 2D and 3D geometric constraint solving. Computer-Aided Design, 38: 1-13, 2006.
- [Ge 1999] Ge J.-X., Chou S.-C. and Gao X.-S. Geometric constraint satisfaction using optimization methods. Computer-Aided Design, 31: 867-879, 1999.
- [Gibbson 1985] Gibbson A. Algorithmic graph theory: Cambridge University Press, 1985.
- [Gilbert 1992] Gilbert J.G., Moler C. and Schreiber R. Sparse matrices in MATLAB: design and implementation. SIAM Journal on Matrix Analysis and Applications, 13: 333-356, 1992.
- [Glachet 1991] Glachet R., Dhome M. and Lapresté J.T. Finding the perspective projection of an axis of revolution. Pattern Recognition Letters, 12: 693-700, 1991.
- [Golub 1996] Golub G.H. and Van L.C.F. Matrix Computations. Baltimore, MD:

Johns Hopkins University Press, 1996.

[Gondran 1982] Gondran M. Graphs and Algorithms. New York, NY, USA: Wiley, 1982.

[Gould 1988] Gould R. Graph Theory. Redwood City, CA: Addison- Wesley, 1988.

[Gray 1997] Gray P., Hart W.E., Painton L., Phillips C., Trahan M. and Wagner J. A survey of global optimization methods.
<http://www.cs.sandia.gov/opt/survey/main.html>, 1997.

[Grimstead 1996] Grimstead I.J. and Martin R.R. Incremental line labeling for sketch input of solid models. Computer Graphics Forum, 15: 155-166, 1996.

[Herbert 1987] Herbert D. Study drawings in architectural design: applications of CAD systems. In: workshop of the association for computer aided design in architecture, 1987.

[Hoffmann 1995a] Hoffmann C.M. and Vermeer P. Geometric constraint solving in R2 and R3. In Du D.Z. and Hwang F. (Eds.), Computing in Euclidean Geometry. Singapore: World Scientific Publishing, 1995a, pp. 266-298.

[Hoffmann 1995b] Hoffmann C.M. and Vermeer P.J. A spatial constraint problem. In: Computational Kinematics Workshop, Nice, France, 83-92, 1995b.

[Hoffmann 1997a] Hoffmann C.M. and Joan-Arinyo R. symbolic constraints in constructive geometric constraint solving. J. Symb. Comput., 23: 287-299, 1997a.

[Hoffmann 2000] Hoffmann C.M. and Yuan B. On spatial constraint solving approaches. In: Proc. ADG 2000, ETH Zurich, 2000.

- [Hoffmann 1997b] Hoffmann C.M., Lomonosov A. and Sitharam M. Finding solvable subsets of constraint graphs. LNCS 1330: 463-477, 1997b.
- [Hoffmann 1998] Hoffmann C.M., Lomonosov A. and Sitharam M. Geometric constraint decomposition. In Bruderlin B. and Roller D. (Eds.), Geometric Constraint Solving and Applications. Berlin: Springer, 1998, pp. 170-195.
- [Hoffmann 2001a] Hoffmann C.M., Lomonosov A. and Sitharam M. Decomposition plans for geometric constraint systems, part I: performance measures for CAD. Journal of Symbolic Computation, 31: 367-408, 2001a.
- [Hoffmann 2001b] Hoffmann C.M., Lomonosov A. and Sitharam M.J. Decomposition plans for geometric constraint problems, part II: new algorithms. Symbolic Computation, 31: 409-427, 2001b.
- [Hoffmann 2004] Hoffmann C.M., Sitharam M. and Yuan B. Making constraint solvers more usable: overconstraint problem. Computer-Aided Design, 36: 377-399, 2004.
- [Hsu 1996] Hsu C.-Y. Graph-based approach for solving geometric constraint problems. PhD thesis. Salt Lake City, UT. University of Utah. 1996.
- [Huffman 1971] Huffman D.A. Impossible objects as nonsense sentences. In Edinburgh B.M. and Michie D. (Eds.), Machine Intelligence. Edinburgh: Edinburgh University Press, 1971, pp. 295-323.
- [Igarashi 2001] Igarashi T. and Hugues J.F. A suggestive interface for 3D drawing. In: 14th Annual Symposium on User Interface Software and Technology, ACM UIST'01, Orlando, FL, 2001.

- [Jenkins 1993] Jenkins D.L. and Martin R.R. The importance of free hand sketching in conceptual design: automatic sketch input. ASME Conference on Design theory and Methodology (DTM'93), 53: 115-128, 1993.
- [Jermann 2002] Jermann C., Neveu B. and Trombettoni G. A new structural rigidity for geometric constraint systems. In: proceedings of the 4th International Workshop on Automated Deduction in Geometry, 87-106, 2002.
- [Jermann 2003] Jermann C., Neveu B. and Trombettoni G. Algorithms for identifying rigid subsystems in geometric constraint systems. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 233-238, 2003.
- [Jiang 1992] Jiang X.Y. and Bunke H. A simple and efficient algorithm for determining the symmetries of polyhedra. Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing, 54: 91-95, 1992.
- [Joan-Arinyo 2003] Joan-Arinyo R., Soto-Riera A., Vila-Marta S. and Vilaplana-Pastó J. Transforming an under-constrained geometric constraint problem into a well-constrained one. In: Proceedings of the eighth ACM symposium on Solid modeling and application, Seattle, Washington, USA, 33-44, 2003.
- [Joan-Arinyo 2004] Joan-Arinyo R., Soto-Riera A., Vila-Marta S. and Vilaplana-Pastó J. Revisiting decomposition analysis of geometric constraint graphs. Computer-Aided Design, 36: 123-140, 2004.
- [Kanade 1981] Kanade T. Recovery of the three-dimensional shape of an object from a single view. Artificial Intelligence, 17: 409 - 460, 1981.

- [Kang 2004] Kang D.J., Masry M. and Lipson H. Reconstruction of a 3D object from a main axis system. In: AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural, 63-69, 2004.
- [Kondo 1992] Kondo K. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design*, 24: 141-147, 1992.
- [Kramer 1992] Kramer G. Solving geometric constraint systems. Cambridge, MA, USA: MIT Press, 1992.
- [Langbein 2002] Langbein F.C., Marshall A.D. and Martin R.R. Numerical methods for beautification of reverse engineered geometric models. In: *Proceedings of the Geometric Modeling and Processing - Theory and Applications* Wako, Japan, 159 - 168, 2002.
- [Langbein 2004] Langbein F.C., Marshall A.D. and Martin R.R. Choosing consistent constraints for beautification of reverse engineered geometric models. *Computer-Aided Design*, 36: 261-278, 2004.
- [Latham 1996] Latham R.S. and Middleditch A.E. Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design*, 28: 917-928, 1996.
- [Leclerc 1992] Leclerc Y.G. and Fiscler M.A. An optimization based approach to the interpretation of single line drawings as 3D wire frame. *Computer Vision*, 9: 113-136, 1992.
- [Lee 1996] Lee J.Y. and Kim K. Geometric reasoning for knowledge-based parametric design using graph representation. *Computer-Aided Design*, 28: 831-941, 1996.
- [Lee 1998] Lee J.Y. and Kim K. A 2D geometric constraint solver using DOF-based

- graph reduction. *Computer-Aided Design*, 30: 883-896, 1998.
- [Lei 1998] Lei Y. and Wong K.C. A novel method for detecting and localising of reflectional and rotational symmetry under weak perspective projection. *International Conference on Pattern Recognition*, 1: 417 - 419, 1998.
- [Leou 1987] Leou J. and Tsai W. Automatic rotational symmetry determination for shape analysis. *Pattern Recognition*, 20: 571-582, 1987.
- [Li 2001] Li Y.T., Hu S.M. and Sun J.G. On the numerical redundancies of geometric constraint systems. In: *Proceedings of Pacific Graphics 2001*, Tokyo, 118-123, 2001.
- [Li 2002] Li Y.T., Hu S.M. and Sun J.G. A constructive approach to solving 3-D geometric constraint systems using dependence analysis. *Computer-Aided Design*, 34: 97-108, 2002.
- [Light 1982] Light R. and Gossard D.C. Modification of geometric models through variational geometry. *Computer-Aided Design*, 14: 209-214, 1982.
- [Lipson 1998] Lipson H. An interface for 3D conceptual design based on freehand sketching. PhD thesis. Israel. Israel Institute of Technology. 1998.
- [Lipson 1996] Lipson H. and Shpitalni M. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design*, 28: 651-663, 1996.
- [Liu 2001] Liu J. and Lee Y.T. A Graph-based method for face identification from a single 2D line drawing. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23: 1089-1106, 2001.

- [Liu 2005] Liu J. and Tang X. Evolutionary search for faces from line drawings. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27, 2005.
- [Liu 2002] Liu J., Lee Y.T. and Cham W.K. Identifying faces in a 2D line drawing representing a manifold object. IEEE Transaction on Pattern Analysis and Machine Intelligence, 24: 1579-1593, 2002.
- [Marill 1991] Marill T. Emulating the human interpretation of line drawings as three-dimensional objects. Computer Vision, 6: 147-161, 1991.
- [Marola 1989] Marola G. On the detection of the axes of symmetry of symmetric and almost symmetric planar images. IEEE Transaction on Pattern Analysis and Machine Intelligence, 11: 104-108, 1989.
- [Martínez 2005] Martínez M.L. and Félez J. A constraint solver to define correctly dimensioned and overdimensioned parts. Computer-Aided Design, 37: 1353-1369, 2005.
- [Masry 2005] Masry M. and Lipson H. A sketch-based interface for iterative design and analysis of 3D objects. In: Proceedings of Eurographics workshop on Sketch-Based Interfaces, Dublin, Ireland, 109-118, 2005.
- [Mathworks 2006] Mathworks. Matlab - The language of technical computing. 2006.
- [Matsuda 1993] Matsuda T., Yamamoto K. and Yamada H. Detection of partial symmetry using correlation with rotated-reflected images. Pattern Recognition, 26: 1245-1253, 1993.
- [Mckay 1981] McKay B.D. Practical Graph Isomorphism. Congressus Numerantium, 30: 45-87, 1981.

- [Mckay 1990] Mckay B.D. Nauty User's Guide (version 1.5), Technical Report TR-CS-90-02. Computer Science Department, Australian National University, 1990.
- [Meeran 1999] Meeran S. and Taib J.M. A generic approach to recognizing isolated, nested and interacting features from 2D drawings. *Computer-Aided Design*, 31: 891-910, 1999.
- [Morgan 2004] Morgan M. Epipolar resampling of linear array scanner scenes. PhD thesis. University of Calgary. 2004.
- [Nalwa 1989] Nalwa V.S. Line-drawing interpretation: bilateral symmetry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11: 1117-1120, 1989.
- [Noort 1997] Noort A. Solving over-constrained geometric models. Master thesis. The Netherlands. Delft University of Technology. 1997.
- [Noort 1998] Noort A., Dohmen M. and Bronsvoort W.F. Solving over- and underconstrained geometric models. In Bruderlin B. and Roller D. (Eds.), *Geometric Constraint Solving and Applications*: Springer-Verlag, 1998, pp. 107-127.
- [O'Mara 2002] O'Mara D.T.J. Automated facial metrolog. PhD thesis. University of Western Australia. 2002.
- [Oh 1988] Oh W.G., Asada M. and Tsuji S. Model-based matching using skewed symmetry information. In: *Proc. Int'l Conf. Pattern Recognition*, 1043-1045, 1988.
- [Oung 2001] Oung J., Sitharam M., Moro B. and Arbree A. *FRONTIER*: fully

enabling geometric constraints for feature-based modeling and assembly.

Symposium on Solid Modeling and Applications: 307-308, 2001.

[Owen 1991] Owen J. Algebraic solution for geometry from dimensional constraints.

In: ACM Symposium on Foundations of Solid Modeling, Austin, TX, 397-407, 1991.

[Paige 1982] Paige C.C. and Saunders M.A. Algorithm 583: LSQR: sparse linear

equations and least squares problems. ACM Transactions on Mathematical Software, 8: 195-209, 1982.

[Pappas 1989] Pappas T. Pappus' theorem & the nine coin puzzle. In The Joy of

Mathematics. San Carlos, CA: Wide World Publishing, 1989.

[Parui 1983] Parui S.K. and Majumder D.D. Symmetry analysis by computer. Pattern

Recognition, 16: 63-67, 1983.

[Piquer 2003] Piquer A., Company P. and Martin R.R. Skewed mirror symmetry in the

3D reconstruction of polyhedral models. Journal of WSCG, 11, 2003.

[Piquer 2004] Piquer A., Martin R.R. and Company P. Skewed mirror symmetry for

depth estimation in 3D line-drawings. Lecture Notes in Computer Science.

Graphics Recognition. Recent Advances and Perspectives, 3088: 142-153, 2004.

[Pugh 1992] Pugh D. Designing solid objects using interactive sketch interpretation.

In: ACM SIGGRAPH-Computer Graphics Special Issue on 1992 Symposium on Interactive 3D Graphics, 117-126, 1992.

[Qin 2000] Qin S.F., Wright D.K. and Jordanov I.N. From on-line sketching to 2D and

3D geometry: a system based on fuzzy knowledge. Computer-Aided Design, 32:

851-866, 2000.

[Rich 1991] Rich E. and Knight K. Artificial intelligence. New York: McGraw-Hill Inc., 1991.

[Roberts 1963] Roberts L.G. Machine perception of three-dimensional solids: Cambridge, MA., 1963.

[Shen 2001] Shen D.G., Ip H.H.S. and Teoh E.K. Robust detection of skewed symmetries by combining local and semi-local affine invariants. Pattern Recognition, 34, 2001.

[Shpitalni 1996] Shpitalni M. and Lipson H. Identification of faces in a 2D line drawing projection of a wire frame object. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18: 1000-1012, 1996.

[Sitharam 2004] Sitharam M. Combinatorial approaches to geometric constraint solving: problems, progress, directions. In Dutta Janardhan S. (Ed.) To appear in AMS-DIMACS book on computer aided design and manufacturing, 2004.

[Sitharam 2006] Sitharam M., Oung J.-J., Zhou Y. and Arbree A. Geometric constraints within feature hierarchies. Computer-Aided Design, 38: 22-38, 2006.

[Skiena 1990] Skiena S. Implementing discrete mathematics: combinatorics and graph theory with mathematica. Redwood City, CA: Addison-Wesley, 1990.

[Sugihara 1984a] Sugihara K. An algebraic approach to shape-from-image problems. Artificial Intelligence, 23: 59-95, 1984a.

[Sugihara 1984b] Sugihara K. A necessary and sufficient condition for a picture to represent a polyhedral scene. IEEE Transactions on Pattern Analysis and

Machine Intelligence, 6: 578-586, 1984b.

[Sugihara 1986] Sugihara K. (Ed.).Machine interpretation of line drawings.
Cambridge: MIT Press, 1986.

[Sugimoto 1994] Sugimoto K. and Tomita F. Detection of skewed-symmetrical shape.
In: Proceedings of the International Conference on Image Processing, Austin,
Texas, 696-700, 1994.

[Sun 1999] Sun C. and Si D. Fast reflectional symmetry detection using orientation
histograms. Real-Time Imaging, 5: 63-74, 1999.

[Sun 2006] Sun Y. Investigation of the processes in the construction of 3D models
from 2D sketches. PhD thesis. Singapore. Nanyang Technological University.
2006.

[Sunde 1987] Sunde G. A CAD system with declarative specification of shape. In:
Proc. of EuroGraphigs Workshop on Intelligent CAD Systems, Noodwijkerhout,
The Netherlands, 1987.

[Suzuki 1990] Suzuki H., Ando H. and Kimura F. Geometric constraints and
reasoning for geometrical CAD systems. Computers & Graphics, 14: 211-224,
1990.

[Varley 2002a] Varley P.A.C. Automatic creation of boundary-representation models
from single line drawings. PhD thesis. Cardiff University. 2002a.

[Varley 2002b] Varley P.A.C. and Martin R.R. Estimating depth from line drawings.
In: Proc. 7th ACM Symposium on Solid Modeling and Applications, SM02,
180--191, 2002b.

- [Varley 2005] Varley P.A.C., Martin R.R. and Suzuki H. Frontal geometry from sketches of engineering objects: is line labelling necessary? *Computer-Aided Design*, 37: 1285-1307, 2005.
- [Verroust 1992] Verroust A., Schonek F. and Roller D. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24: 531-540, 1992.
- [Waltz 1972] Waltz D.M. Generating semantic descriptions from drawings of scenes with shadows. Tech Rept AI-TR-271. Cambridge USA: M.I.T., 1972.
- [Wang 1991] Wang W. Regular curved object's CSG-rep reconstruction from a single 2D line drawing. In: *Proc.of 1991 SPIE Intelligent Robots and Computer Vision X: Neural, Biological and 3-D Methods*, Boston, USA, 1991.
- [Wang 1989] Wang W. and Grinstein G. A polyhedral object's CSG-Rep reconstruction from a single 2D line drawing. In: *Proc. Of 1989 SPIE Intelligent Robots and Computer Vision III: Algorithms and Techniques*, 230-238, 1989.
- [Weinberg 1966] Weinberg L. A simple and efficient algorithm for determining isomorphisms of planar triply connected graphs. *IEEE Transactions on Circuit Theory*, 13: 142-148, 1966.
- [Werghi 1999] Werghi N., Fisher R., Robertson C. and Ashbrook A. object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design*, 363-399, 1999.
- [Wilczkowiak 2003] Wilczkowiak M., Trombettoni G., Jermann C., Sturm P.F. and Edmond B. Scene modeling based on constraint system decomposition

- techniques. In: International Conference on Computer Vision, Nice, France, 1004-1010, 2003.
- [Wu 1994] Wu W.-T. Mechanical theorem proving in geometries. New York, USA: Springer-Verlag Inc., 1994.
- [Xu 1992] Xu G., Tanaka H.T. and Tsuji S. Right straight homogeneous generalized cylinders with symmetric cross-sections: recovery of pose and shape from image contours. IEEE Conference on Computer Vision and Pattern Recognition: 692-694, 1992.
- [Yip 1999] Yip R.K.K. A Hough transform technique for the detection of parallel projected rotational symmetry. Pattern Recognition Letters, 20: 991-1004, 1999.
- [Yip 2000] Yip R.K.K. A Hough transform technique for the detection of reflectional symmetry and skew-symmetry. Pattern Recognition Letters, 21, 2000.
- [Yip 1994] Yip R.K.K., Tam P.K.S. and Leung D.N.K. Application of elliptic Fourier descriptors to symmetry detection under parallel projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16: 277-286, 1994.
- [Zabrodsky 1993] Zabrodsky H. Computational aspects of pattern characterization - continuous symmetry. PhD thesis. Jerusalem, Israel. Hebrew University. 1993.
- [Zabrodsky 1994] Zabrodsky H. and Weinshall D. 3D symmetry from 2D data. In: Proc. European Conf. Computer Vision, Stockholm, Sweden, 1994.
- [Zabrodsky 1992] Zabrodsky H., Peleg S. and Avnir D. Hierarchical Symmetry. In: Proc. Int'l Conf. Pattern Recognition, The Hague, The Netherlands, 9-11, 1992.
- [Zabrodsky 1995] Zabrodsky H., Peleg S. and Avnir D. Symmetry as a Continuous

Feature. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17:
1154-1166, 1995.

[Zelevnik 1996] Zelevnik R.C., Herndon K.P. and Hughes J.F. SKETCH: An interface
for sketching 3D scenes. In: SIGGRAPH '96, Proceedings of the 23rd Annual
Conference on Computer Graphics, New Orleans, LA, USA, 163-170, 1996.

[Zou 2005] Zou H.L. and Lee Y.T. Detecting minimum over-constrained subgraphs in
2D and 3D based on degree of freedom analysis. Computer-Aided Design and
Applications, 2: 393-402, 2005.