Geometric design of flat layout of 3D folded structures

Liu, Wei

2005

Liu, W. (2005). Geometric design of flat layout of 3D folded structures. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/6064>

<https://doi.org/10.32657/10356/6064>

Nanyang Technological University

Downloaded on 26 Apr 2023 23:32:24 SGT

Geometric Design of Flat Layout of 3D Folded Structures

Liu Wei

School of Mechanical and Aerospace Engineering

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

2005

To Xiaolei, my parents and all other family members

Acknowledgement

The author would like to express his utmost appreciation to Associate Professor Tai Kang for his patient supervision and kindness in all aspects of the author's research work. The author would also like to thank Nanyang Technological University for the research scholarship and all the staffs and students in Design Research Center (DRC) for the best working environment and atmosphere that given to him. Last, but of course not least, the author would like to thank his family for all the mental supports, which is vital for the fulfillment of this report.

Contents

Acknowledgement	I
Contents	II
List of Figures	VI
List of Tables	IX
List of Algorithms	X
Abstract	XI
1 Introduction	1
1.1 Background	1
1.2 Objective	3
1.3 Scope	4
1.4 Original Contributions Arising from This Work	5
1.5 Organization of Thesis	5
2 Geometric Modeling and Graph Theoretic Preliminaries	7
2.1 Structure Classification	7
2.1.1 Manifold Structures	8
2.1.2 Non-manifold Structures	10
2.2 Geometric Modeling	12
2.2.1 Relationships Among Geometric Entities of Structures	12
2.2.2 Solid Model Construction	18
2.2.3 Graph Representation of 3D models	23
2.3 Graph Theory	24
2.3.1 Basic Concepts	24
2.3.2 Graph Representation	24

CONTENTS	III
2.3.3 Graph Traversal	26
2.3.4 Spanning Trees	29
2.3.5 Spanning Tree Enumeration	30
3 Literature Review	33
3.1 Study of Origami	33
3.1.1 Unfolding Types	34
3.1.2 Unfoldability	35
3.1.3 Unfolding Methodologies in Origami	37
3.2 Study of Sheet Metal Design and Thin-walled Objects	41
3.2.1 Unfolding Methodologies in Sheet Metal Design and Thin-Walled Objects	41
3.3 Other Related Work	47
4 Enumeration of Flat Layouts	49
4.1 Modeling of the structures	49
4.1.1 Geometric Modeling	50
4.1.2 Topological Modeling	51
4.2 Flat Layout Generation	55
4.2.1 Creation of Graph Representation From B-rep Models	56
4.2.2 Spanning Tree Enumeration on FAGs	58
4.2.3 Geometric Construction of Flat Layouts	58
4.2.4 Orientation of Faces	63
4.2.5 Overlapping Detection	64
4.2.6 Methodology Outline	68
4.2.7 Time Complexity Analysis of Methodology	69
4.2.8 Methodology Illustration	70
4.3 Other Aspects of Unfolding	71
4.3.1 Depression, Protrusion and Through-hole Features	71
4.3.2 Face Merging Problem	72
4.3.3 Optimality Criteria	73
5 Unfolding with Topological Recognition	78
5.1 Hyper-Common Edge	78
5.1.1 Definitions	78
5.1.2 Problems Arising from Hyper-Common Edges	79
5.1.3 Solution to Hyper-Common Edges	81

CONTENTS	IV
5.2 Topological Recognition with Overlapping Paths	84
5.2.1 Inadmissible Links	85
5.2.2 Exclusive Link Sets	87
5.3 Methodology with Topological Recognition	90
5.3.1 Methodology Outline	90
5.3.2 Time Complexity Analysis	92
6 Improving Efficiency by Subdividing the FAG	94
6.1 Subdividing FAG by Exclusive Link Sets	95
6.1.1 Basic Idea	95
6.1.2 Advantages of Subdividing FAGs	97
6.2 Converting Hyper-Common Edge Sequences to Exclusive Link Sets . .	99
6.3 Sub-FAG Set Construction	101
6.4 Repeated Spanning Trees	101
6.4.1 Sub-FAG Overlapping Problem	102
6.4.2 Indexing Spanning Trees	105
6.4.3 Detecting Repeated Spanning Trees	106
6.5 Overall Methodology	107
6.5.1 Methodology Outline	107
6.5.2 Time Complexity Analysis	109
6.5.3 Space Complexity Analysis	109
7 Results	110
7.1 Example 1	111
7.2 Example 2	112
7.3 Example 3	113
7.4 Example 4	117
7.5 Example 5	119
7.6 Example 6	122
7.7 Example 7	125
7.8 Example 8	128
7.9 Example 9	130
7.10 Example 10	132
7.11 Example 11	134
7.12 Example 12	136
7.13 Example 13	138

7.14 Comparison of the Three Versions	140
8 Conclusion and Recommendations	141
8.1 Conclusion	141
8.2 Recommendations	143
8.2.1 Efficiency Improvement	143
8.2.2 Handling of Unmanageable Structures	144
8.2.3 Intersecting Faces	145
8.2.4 Design for Folding	146
Appendix	147
A Layout Enumeration Tool (LET)	147
A.1 Version 1 System Description	147
A.1.1 I/O Subsystem	147
A.1.2 Preprocessing Subsystem	148
A.1.3 Flat Layout Enumeration Subsystem	149
A.1.4 Entity Pool	149
A.2 Version 2 System Description	149
A.2.1 Preprocessing Subsystem	149
A.2.2 Flat Layout Enumeration Subsystem	149
A.3 Version 3 System Description	150
A.3.1 Preprocessing Subsystem	150
A.3.2 Flat Layout Enumeration Subsystem	151
A.3.3 Main User Interface Screenshot	151
B Computational Geometry for Overlapping Detection	153
B.1 Point Location Search	153
B.2 Intersection Detection Within A Group of Line Segments	154
References	158

List of Figures

1.1	Project objective illustration	3
2.1	Examples of simple polyhedra	9
2.2	An example of a non-simple polyhedron	9
2.3	A non-polyhedral manifold structure	10
2.4	Examples of non-manifold structures	11
2.5	Examples to illustrate relationships among entities	13
2.6	Counting G_{nm} (adapted from Lipson & Shpitalni, 1998)	17
2.7	CSG representation (adapted from Mortenson, 1997)	19
2.8	B-rep Representation (adapted from Mäntylä, 1988)	20
2.9	A sample object for illustration of B-rep structures (adapted from Mäntylä, 1988)	21
2.10	The full winged-edge data structure (adapted from Mäntylä, 1988) . . .	22
2.11	Examples of FAG representation	23
2.12	Graphs	25
2.13	Adjacency matrix	25
2.14	Packed-adjacency list	26
2.15	Linked-adjacency list	27
2.16	BFS and DFS	28
2.17	Graph with 4 nodes and 5 links	29
2.18	Spanning trees	30
2.19	Deriving a new spanning tree from a current tree T	30
3.1	An example of ununfoldable structures	35
3.2	Probabilities of overlapped unfoldings of randomly generated polytopes (adapted from O'Rourke, 1998)	36
3.3	Shortest path and ridge tree on a pyramid. (adapted from Agarwal et al., 1997)	38
3.4	Star unfolding for the pyramid (adapted from Agarwal et al., 1997) . . .	38

LIST OF FIGURES

VII

3.5	Source unfolding for the pyramid (adapted from Agarwal et al., 1997)	39
3.6	An open box and its graph representation. (adapted from Shpitalni & Lipson, 2000)	42
3.7	Seam line styles (adapted from Lin & Yang, 1998)	44
4.1	Illustration of applied B-rep data structure	50
4.2	An example to show data structure	51
4.3	Full data structure for example in Figure 4.2	52
4.4	Different objects with same FAG representation	54
4.5	B-rep data (loop table) for the structures shown in Figure 4.4	54
4.6	Different common edges shared by same faces	55
4.7	Flat layout generation	59
4.8	An example of add face algorithm	60
4.9	Relative reference of face	60
4.10	Relative coordinates of vertices of face $abef$	61
4.11	Final results after coordinate transform	62
4.12	Orientation of faces	64
4.13	Selection of the normal vectors for faces	65
4.14	Possible relative positions of faces	65
4.15	Possible relative positions of intersecting lines	66
4.16	Face relationships when sharing a common edge	67
4.17	Two special cases for intersection style (c)	68
4.18	Depression treatment	72
4.19	The face merging problem	73
4.20	Optimality criteria example	74
4.21	Different optimal results based on different optimality criteria.	77
5.1	Incomplete splitting of a hyper-common edge	79
5.2	Incorrect splitting of a hyper-common edge	80
5.3	Faces of a hyper-common edge set.	82
5.4	Orientation of normal vectors of the hyper-common edge set.	82
5.5	Valid and invalid splitting of hyper-common edges	84
5.6	Topologically valid spanning tree.	84
5.7	Inadmissible links	86
5.8	Illustration of overlapping paths.	88
5.9	An example of overlapping path with distance of 7.	89

LIST OF FIGURES

VIII

6.1	Structures with high percentage of topologically invalid spanning trees .	95
6.2	Illustration of sub-FAGs	96
6.3	Constructing subgraphs with no exclusive links.	98
6.4	Convert hyper-common edge sequences to exclusive link sets	100
6.5	Some structures for repeated spanning tree analysis	104
6.6	Indexing spanning trees	106
8.1	An unmanageable structure	144
8.2	Intersecting Faces	145
8.3	Addition of flaps and allowance for material thickness	146
A.1	LET System: version 1	148
A.2	LET System: version 2	150
A.3	LET System: version 3	151
A.4	LET main user interface screenshot	152
B.1	Point location query	154
B.2	Search for intersections using sweep-line technique	155
B.3	Intersection detection between two line segments	156

List of Tables

2.1	Differences between manifold and non-manifold structures	11
2.2	The number of main entities of the examples	13
2.3	Time complexity analysis	27
4.1	Methodology Illustration	70
6.1	Results for the structures shown in Figure 6.1	95
6.2	Repeated spanning tree analysis	105
7.1	Example 1	111
7.2	Example 2	112
7.3	Example 3	113
7.4	Example 4	117
7.5	Example 5	119
7.6	Example 6	122
7.7	Example 7	125
7.8	Example 8	128
7.9	Example 9	130
7.10	Example 10	132
7.11	Example 11	134
7.12	Example 12	136
7.13	Example 13	138
7.14	CPU Time Comparison of the Three Versions of LET	140
B.1	Sorted edges	155

List of Algorithms

4.1	Face connectivity algorithm	57
4.2	Edge connectivity algorithm	57
4.3	The add face algorithm	62
4.4	The unfolding methodology based on spanning tree enumeration	69
5.1	Modified Unfolding Methodology	91
6.1	Construct Sub-FAG Set E_G	101
6.2	SearchSubFAG (G, S_l, l_i)	102
6.3	Subdividing FAG methodology	108

Abstract

Due to environmental considerations, corrugated paperboards folded into appropriate 3D structural shapes are increasingly being used as packaging cushions, as a substitute for those traditionally made of polymer foams. However, since paperboards are manufactured in the form of sheets, 3D structures have to be created from these boards by folding. The design of the necessary flat layout pattern of a board that can be folded into a reasonably complex and intricate shape is a process requiring a lot of costly trial-and-error and creativity on the part of the designer. This thesis describes three progressively enhanced methodologies that were developed for unfolding 3D folded structures so as to generate the required flat layouts. The key to the first methodology is a computer representation of the topology/connectivity of the faces of the 3D folded structure by a graph-theoretic model, and an algorithm to operate on this model to unfold and enumerate the topology and geometry of the planar layout. Building upon the first methodology, a topological recognition method is proposed in the second methodology for handling more complex structures (like those containing common edges that are shared by more than two faces) and improving enumeration efficiency. A further improvement in the third methodology divides the overall enumeration problem into subproblems in a way that systematically bypasses invalid results. Some other complications and issues related to unfolding (such as structural features with depressions, protrusions and through holes, face merging problems, and generating optimal layouts) have also been addressed in this thesis. The methodologies have been implemented in a C++ program and a series of example problems have been successfully solved to illustrate and verify the methodologies.

CHAPTER 1

Introduction

1.1 Background

In the packaging industry, buffers are commonly used to protect the contents in the package from damage that may occur during the transportation/distribution process. These buffers are also sometimes referred to as packaging cushions. Currently, most of these cushions or buffers are made of plastics, usually in the form of polymer foams. These foams have been preferred over other materials in the packaging industry for several reasons, namely availability, low cost, ease of forming into various sizes and shapes, and low weight. However, the fact that plastics are non-biodegradable has led to concerns of its effect on waste management and the environment. Plastic foam is also difficult to recycle due to the problem of contamination. The recycling of these materials requires advanced and expensive process equipment, which deemed it uneconomical since the cost of these plastics is low and it is readily available.

Significant amount of work has been done to address this problem (Brody & Marsh, 1997). The search for a more environment-friendly substitute for polymers has been

ongoing for some years now and a few alternatives have met with some level of commercial success and acceptance, and these are (molded) paper pulp, paperboard or corrugated board (folded into 3D structures/partitions), and starch-based foam. Each of these alternatives has its own advantages and shortcomings, although all share the same disadvantage of higher costs compared to polymer foams. Relative to the other two alternatives, the main advantage of folded paperboard or corrugated board is that the production of the boards themselves is based on well-established manufacturing processes already familiar to the packaging industry. Relative to polymer foam, the use of paper for packaging is deemed more environmentally responsible since it is by nature biodegradable, recyclable, re-usable and is derived from a renewable (organic) source. However, the disadvantages of folded boards are that, firstly, the folding is usually a labor-intensive process often done manually and seldom where automation is economically viable. Secondly, the geometric design of a flat layout (of paperboard) that can be folded into some desired 3D folded structure is a challenging problem that requires creativity/ingenuity, experience and much trial-and-error on the part of the designer. As paperboards or corrugated boards are manufactured as flat sheets, a 3D board structure (with some complex shape to hold/protect the product being packaged and absorb/cushion any force impact) has to be produced by folding from a suitable flat paperboard pattern/layout. Different products with varying geometry and complexity will require much innovation from the designer to create various 3D folded packaging structures and the corresponding flat layout.

Although it is oriented towards the task of automatically developing flat layouts of paperboard buffers in package design, the unfolding methodology implemented in this research has many other potential applications as well. The methodology is relevant to sheet metal product design and origami, e.g. unfolding a given 3D sheet metal product model for manufacturing or unfolding a given polytope. For the reason that we deal with some kinds of non-simple polyhedra and non-manifold objects, the methodology can also serve as a research tool in geometric study of these relatively unstudied fields.

1.2 Objective

Given a product to be packaged, the normal procedure to develop buffers for it is to firstly create 3D models of the buffers and then try to unfold/flatten it. The objective here is to automatically produce the flat layout of a given 3D buffer model which has been input into the computer by some means, e.g. via a CAD model. There may be multiple solutions for a given structure, i.e. there may be more than one 2D flat layout for a given 3D folded structure. All solutions are to be enumerated so as to choose one or more optimal layouts that satisfy certain optimality criteria. For simplicity, we use a simple L-shape structure shown in Figure 1.1 to illustrate our objective.

Figure 1.1. Project objective illustration

The focus of this project is the development of a methodology that can deal with a wide range of structural topologies. As 3D objects are involved, a suitable mathematical

model is needed to represent such objects. Exhaustive enumeration is desired so as to generate all possible non-overlapping layouts. A way to flatten a given 3D folded structure is necessary to output graphically the flat layouts. Moreover, there are some problems to be addressed before achieving the goal. For example, modifications of 3D folded structures and detecting overlapping of the faces of a flat layout.

1.3 Scope

The project will look into various aspects in the development of a methodology to automate the process of developing 2D flat layouts of a given 3D folded structure.

The structures to be dealt with will have the following characteristics:

- The structure is constructed by piece-wise flat surfaces, each bounded by straight edges (i.e. polygons).
- Each structure contains only one shell (i.e. one component).
- Since the structure is a real physical object, there are no dangling edges, isolated edges or isolated points.

In order to represent and handle structures through computers, the B-rep method in geometric modeling is applied. However, because a basic B-rep implementation may not offer efficient access to certain topological data such as face adjacency (which is vital to the unfolding process), thus a graph representation technique is also applied to address this problem. This leads to the requirement of some concepts and techniques used in graph theory and topology.

It is not enough if only topologically valid layouts are obtained, i.e. theoretically correct layouts according to the topological data, which may not be valid layouts according to geometric attributes since some faces may overlap. Structures need to be actually unfolded. Some knowledge about analytic geometry and computational geometry are needed for this purpose.

1.4 Original Contributions Arising from This Work

Firstly, the methodology developed in this research extended the range of problem structures that have been treated by other researchers. It aims at providing a generic method for unfolding both manifold and non-manifold structures. Complications like having more than two faces sharing one common edge (which is named as a *hyper-common edge*) was considered and addressed in this work. Structures containing this problematic feature may be unfolded by other methods without overlapping, but the unfolded flat layouts may be erroneous and cannot be folded back into the original 3D structure (see Section 5.1.2).

Secondly, the concept of overlapping path was proposed in this work. By using overlapping paths and topological recognition within the graph representation of a 3D folded structure, a heuristic unfolding methodology (see Chapter 5) was developed to accelerate the overall unfolding efficiency. Efficiency improvement is achieved by recognizing overlapped faces topologically without having to construct the flat layout geometrically.

Thirdly, a more efficient methodology was proposed in Chapter 6. It divides the whole problem into a group of subproblems and removes all foreknown invalid results in the subdividing stage. In this way, efficiency is improved because a large number of invalid or overlapping layouts will not even be topologically enumerated in the first place. Subproblems are relatively independent of one another, and therefore distributed computation may be also used for handling these subproblems.

1.5 Organization of Thesis

In Chapter 2, some key theories that form the basis of our methodology are discussed, i.e. some theoretical concepts and methodologies from geometric modeling and graph theory. Several related research fields such as origami and sheet metal product design are reviewed in Chapter 3 so as to build up a holistic appreciation of our work. A basic methodology for flattening folded structures will be explained in Chapter 4. Chapters 5 and 6 will explain how the unfolding methodologies are extended and enhanced for

handling more complex structures and for improving the overall efficiency. Case studies are performed and results will be shown in Chapter 7. Finally, conclusion and recommendations are given in Chapter 8.

CHAPTER 2

Geometric Modeling and Graph Theoretic Preliminaries

In this chapter, several key theories applied in the unfolding methodology will be discussed. Firstly, different types of structures are analyzed to make clear which are of interest. Secondly, how structural data should be stored and can be easily understood by computers will be discussed, which is a matter of geometric modeling of 3D models. And then some concepts and algorithms from graph theory that are important to the unfolding procedures will be described.

2.1 Structure Classification

To make the objectives clear, it is useful to study structure types before trying to unfold them. Structures can be broadly classified into two major categories: *Manifolds* and *Non-manifolds*. Manifold structures can be further classified into *Simple Polyhedra*, *Non-simple Polyhedra* and *Non-polyhedral Manifold Structures*.

2.1.1 Manifold Structures

By definition, a *manifold* structure M is a topological space where every point has a neighborhood topologically equivalent to an open disk of E^2 , where E^2 is the two-dimensional Euclid's Space (Mäntylä, 1988). Intuitively, a bug living on M sees a continuous simple region of the surface all around itself. This definition asserts that each manifold structure requires that (1) all edges separate exactly two faces and (2) all vertices are surrounded by a single circuit of faces.

Normally, manifold structures are primarily constructed by *piecewise flat surfaces*, which are defined as surfaces formed by taking a collection of planar pieces and joining them together along their edges. Any surface formed in this way will obviously be flat everywhere except possibly along the edges.

Polyhedra

According to definition, a polyhedron is without any doubt in the scope of manifold structures. A *polyhedron* means an arrangement of polygons such that two and only two polygons meet at an edge, and it is possible to traverse the surface of the polyhedron by crossing its edges and moving from one polygonal face to another until all polygons have been traversed by this continuous path (Mortenson, 1997).

Simple Polyhedra

The term *simple polyhedron* refers to a polyhedron that can be continuously deformed into a sphere, i.e. if it is inflated, it can become a sphere (Mortenson, 1997). Furthermore, the term *convex* is applied to every polyhedron that lies entirely on one side of each of its polygonal faces (i.e. a polyhedron in which every intersection angle of adjacent polygonal surfaces is equal to or larger than π), while non-convex simple polyhedra will break this criterion. Figure 2.1 depicts some examples of simple polyhedra.

As we can see, Figure 2.1a is a convex simple polyhedron according to the criterion of convexity while Figure 2.1b is a non-convex simple polyhedron. Normally, simple polyhedra may be unfolded without any difficulty. Unfortunately, most structures are

Figure 2.1. Examples of simple polyhedra. (a) is a convex simple polyhedron and (b) is a non-convex simple polyhedron.

not simple polyhedra.

Non-simple Polyhedra

Another category of polyhedra is the *non-simple polyhedron*. It is topological equivalents of any solid object with holes in it. Figure 2.2 shows an example of a non-simple polyhedron. It is a cube with a through hole. By and large, non-simple polyhedra cannot be unfolded directly, i.e. the structures of non-simple polyhedra may need to be modified before being flattened.

Figure 2.2. An example of a non-simple polyhedron

Non-polyhedral Manifold Structures

Polyhedra are only subsets of manifold structures. The structure shown in Figure 2.3 is a cube with an internal hole. It can not be called a polyhedron because it breaks the criterion that it should be possible to traverse the surface of the polyhedron by crossing its edges and moving from one polygonal face to another until all polygons have been tra-

versed by this continuous path. But it is a manifold structure according to the definition of manifold.

Figure 2.3. A non-polyhedral manifold structure

2.1.2 Non-manifold Structures

The term “*non-manifold*” is used because non-manifold structures relax some of the topological correctness criteria for manifold structures. Recall that manifold structures require that (1) all edges separate exactly two faces and (2) all vertices are surrounded by a single circuit of faces. This does not hold for many structures that contain boundary edges, dangling faces, etc. Figure 2.4 shows some non-manifold structures.

All examples shown in Figure 2.4 contain problem entities that break at least one of the two criteria for manifold structures. In Figure 2.4a, there are several boundary edges that are adjacent to only one face. Figure 2.4b shows an example where edges in non-manifold structures may have more than two adjacent faces. The problem vertex in Figure 2.4c is surrounded by more than one circuit of faces. Table 2.1 shows the differences between manifold and non-manifold structures that are constructed by vertices, edges and faces. In fact, manifold structures are special cases of non-manifold structures according to their definitions. However, manifold structures are excluded from non-manifolds here so as to make the concepts clear. Non-manifold structures in turn refer to those structures that are not manifold.

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 11

Figure 2.4. Examples of non-manifold structures**Table 2.1.** Differences between manifold and non-manifold structures

Geometric elements	Manifold structures	Non-manifold structures
Faces	are parts of structure surface	can be parts of structure surface, interior or even separated from the structure
Edges	have exactly two adjacent faces	can have any number of adjacent faces, including none at all
Vertices	adjacent to at least three faces (or edges)	adjacent to any number of faces (or edges), and also can be intersection points of objects, faces, edges, and can even be isolated points.

2.2 Geometric Modeling

2.2.1 Relationships Among Geometric Entities of Structures

To check the validity of a given structure or a flattened layout, it is necessary to know the relationships among the geometric entities, e.g. vertices, edges and faces, etc. Following from the previous subsection, there are some well-known relationships that help to address this need.

Euler's Formula

There is a well-known relationship between the number of vertices, edges and faces of some kinds polyhedra, called Euler's formula. Let V , E and F represent the number of vertices, edges and faces of a polyhedron, respectively. Euler's formula states that:

$$V - E + F = 2 \quad (2.1)$$

However, not all polyhedra satisfy Euler's formula. Certain conditions must be satisfied before this formula can be applied to a given object (Mortenson, 1997):

- All the faces must be simply connected, with no holes in them, and bounded by a single loop of edges, i.e. they are topological disks.
- The solid object must be simply connected (i.e. its complement is connected) and has no holes through it.
- Each edge must adjoin exactly two faces and terminate at a vertex at each end.
- At least three edges meet at each vertex.

Those objects that violate any of the above conditions may or may not satisfy Euler's formula. Examples shown in Figure 2.5 are used to illustrate Euler's formula as well as the formulas that will be discussed in later sections. Figure 2.5a is a simple polyhedron while Figure 2.5b and c are two non-simple polyhedra. Recalling the earlier discussion, Figure 2.5d is a non-polyhedral structure. Figure 2.5e and f are two non-manifold objects. The number of main entities of these structures are shown in Table 2.2.

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 13

Note :

e_1, e_2, e_3, e_4 and e_5, e_6, e_7, e_8 and
 $e_9, e_{10}, e_{11}, e_{12}$ form three openings

Figure 2.5. Examples to illustrate relationships among entities

Table 2.2. The number of main entities of the examples

Example	Vertices	Edges	Faces
(a)	12	18	8
(b)	16	24	11
(c)	16	24	10
(d)	16	24	12
(e)	12	18	5
(f)	10	15	7

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 14

Thus for

$$\begin{aligned} \text{(a)} \quad V - E + F &= 12 - 18 + 8 = 2 & \text{(b)} \quad V - E + F &= 16 - 24 + 11 = 3 \\ \text{(c)} \quad V - E + F &= 16 - 24 + 10 = 2 & \text{(d)} \quad V - E + F &= 16 - 24 + 12 = 4 \\ \text{(e)} \quad V - E + F &= 12 - 18 + 5 = -1 & \text{(f)} \quad V - E + F &= 10 - 15 + 7 = 2 \end{aligned}$$

From the results, it can be concluded that Euler's formula may or may not be satisfied by non-simple polyhedra or non-manifold structures.

Modified Euler's Formula for Non-simple Polyhedron

Non-simple polyhedra are different from simple ones in that they contain through holes. In general, Euler's formula cannot be applied for non-simple polyhedra, for they violate at least the second condition. Thus it is modified for the non-simple polyhedra (Mortenson, 1997), and the modified Euler's formula states that:

$$V - E + F - H + 2P = 2B \quad (2.2)$$

where V , E and F retain their meaning as in the original Euler's formula; H denotes the number of holes in faces; P denotes the number of passages (holes through the entire object); and B denotes the number of separate, disjoint bodies (objects). Note that this modified Euler's formula can also be applied for simple polyhedra.

In order to illustrate this formula, another three parameters, i.e. H , P and B for the examples are needed. For (a) $H = 0$, $P = 0$ and $B = 1$; (b) $H = 1$, $P = 0$ and $B = 1$; (c) $H = 2$, $P = 1$ and $B = 1$; H , P and B for (d), (e) and (f) are the same: $H = 0$, $P = 0$ and $B = 1$. Thus the results of applying Equation 2.2 to these six examples are:

$$\begin{aligned} \text{(a)} \quad V - E + F - H + 2P &= 12 - 18 + 8 - 0 + 0 = 2 = 2B \\ \text{(b)} \quad V - E + F - H + 2P &= 16 - 24 + 11 - 1 + 0 = 2 = 2B \\ \text{(c)} \quad V - E + F - H + 2P &= 16 - 24 + 10 - 2 + 2 = 2 = 2B \\ \text{(d)} \quad V - E + F - H + 2P &= 16 - 24 + 12 - 0 + 0 = 4 \neq 2B \\ \text{(e)} \quad V - E + F - H + 2P &= 12 - 18 + 5 - 0 + 0 = -1 \neq 2B \\ \text{(f)} \quad V - E + F - H + 2P &= 10 - 15 + 7 - 0 + 0 = 2 = 2B \end{aligned}$$

The results show that Equation 2.2 is satisfied for simple polyhedra and non-simple

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 15

polyhedra, but cannot be applied for non-polyhedral manifold structures and non-manifold structures.

Euler-Poincaré Formula

All manifold structures have an invariant rule in the relationship between their number of vertices, edges, and faces, known as the Euler-Poincaré formula (Mäntylä, 1988), which states that:

$$V - E + F = 2(S - G) \quad (2.3)$$

However, it does not apply to the structures that have rings (i.e. a ring is an inner loop, i.e. a collection of connected edges that form a hole within a face). Thus it is extended to the following form (Shene, 2002):

$$V - E + F - R - 2(S - G) = 0 \quad (2.4)$$

where V , E and F denotes the number of vertices, edges and faces of the structure, respectively. R denotes the number of rings. S denotes the number of shells, which is defined as a collection of connected faces each of which shares at least an edge with another face in this set. A shell is closed if it is topologically continuous and bounds a topological volume, i.e. there does not exist a path from any point outside the shell to any point inside, without penetrating the shell itself. In the case of manifold objects, all shells are closed. G is called genus, which is defined as the maximum number of non-intersecting loops that can be drawn on the surface before dividing it into two or more regions.

Returning to the examples shown in Figure 2.5, L , S and G for each structure are: (a) $R = 0$, $S = 1$ and $G = 0$; (b) $R = 1$ (located in the top face), $S = 1$ and $G = 0$; (c) $R = 2$ (one in the top face and one in the bottom face), $S = 1$ and $G = 1$; (d) $R = 0$, $S = 2$ and $G = 0$; (e) $R = 0$, $S = 1$ and $G = 0$; (f) $R = 0$, $S = 1$ and $G = 0$. Note that for non-manifold structures, G is not easy to count. This is because faces for non-manifold structures are always looked upon as two-sided meeting at free boundary edges; while for manifold structures, faces only have one side.

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 16

All necessary parameters for Equation 2.4 are known, so results can be listed as follows:

- (a) $V - E + F - R - 2(S - G) = 12 - 18 + 8 - 0 - 2(1 - 0) = 0$
- (b) $V - E + F - R - 2(S - G) = 16 - 24 + 11 - 1 - 2(1 - 0) = 0$
- (c) $V - E + F - R - 2(S - G) = 16 - 24 + 10 - 2 - 2(1 - 1) = 0$
- (d) $V - E + F - R - 2(S - G) = 16 - 24 + 12 - 0 - 2(2 - 0) = 0$
- (e) $V - E + F - R - 2(S - G) = 12 - 18 + 5 - 0 - 2(1 - G) < 0$
- (f) $V - E + F - R - 2(S - G) = 10 - 15 + 7 - 0 - 2(1 - 0) = 0$

For the example (e), the Euler-Poincaré formula is not satisfied, no matter how G is counted. The results illustrate that the Euler-Poincaré formula may or may not be satisfied by non-manifold structures. This constrains the use of the Euler-Poincaré formula exclusively to manifold structures.

Modified Euler's Formula for Non-manifold Structures

Lipson & Shpitalni (1998) proposed a modified Euler formula for non-manifold geometry such as general sheet models:

$$V - E + F = S + M - G_{nm} \quad (2.5)$$

where V , E and F have the same meanings as previously defined. S denotes the number of shells and M denotes the number of closed volumes, which correspond to closed surfaces from which no curve can be cast to the exterior (infinity or another exterior surface). G_{nm} is the sum of non-manifold genus for each single component, which is defined as the maximum number of non-intersecting closed curves that can be drawn on the part surface before partitioning its surface into two previously connected disconnected regions. Note that for the purpose of this definition, a surface is considered two-sided, and the sides meet at free edges. This leads to the different methods in counting G_{nm} for non-manifold structures and G for manifold structures. For example, a torus has genus 1 under the classical manifold definition of genus. However, the genus of a torus is 2 in the non-manifold sense because two closed curves can be drawn on its surface, one

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 17

on the internal side and the other on the external side, and still no previously connected surfaces will be partitioned. Figure 2.6 illustrates how G_{nm} is counted for the example (e).

Equation 2.5 is proposed to handle all non-manifold structures (including manifold objects). However, it seems inapplicable for some structures, e.g. example (b). This structure has 16 vertices, 24 edges, 11 faces, 1 shell, 1 volume and G_{nm} is 0. Thus the left hand side of Equation 2.5 is $V - E + F = 16 - 24 + 11 = 3$; while the right hand side is $S + M - G_{nm} = 1 + 1 - 0 = 2$, which means Equation 2.5 is not satisfied in this case.

Figure 2.6. Counting G_{nm} (adapted from Lipson & Shpitalni, 1998)

Based on the Euler-Poincaré formula for manifold objects, an invariant for all non-manifold structures (and manifolds) can be obtained if another parameter (i.e. the number of rings R) is added to Equation 2.5. The result is the following equation:

$$V - E + F - R = S + M - G_{nm} \quad (2.6)$$

where V, E, F, S, M, G_{nm} retain their meanings as in Equation 2.5, and R is the number of rings.

Examples shown in Figure 2.5 are still used here to illustrate Equation 2.6. All other necessary parameters for them are: (a) $M = 1$ and $G_{nm} = 0$; (b) $M = 1$ and $G_{nm} = 0$; (c) $M = 1$ and $G_{nm} = 2$; (d) $M = 2$ and $G_{nm} = 0$; (e) $M = 0$ and $G_{nm} = 2$ (f) $M = 1$ and $G_{nm} = 0$. Thus the results are:

$$(a) V - E + F - R = 12 - 18 + 8 - 0 = 2 = 1 + 1 - 0 = S + M - G_{nm}$$

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 18

$$(b) V - E + F - R = 16 - 24 + 11 - 1 = 2 = 1 + 1 - 0 = S + M - G_{nm}$$

$$(c) V - E + F - R = 16 - 24 + 10 - 2 = 0 = 1 + 1 - 2 = S + M - G_{nm}$$

$$(d) V - E + F - R = 16 - 24 + 12 - 0 = 4 = 2 + 2 - 0 = S + M - G_{nm}$$

$$(e) V - E + F - R = 12 - 18 + 5 - 0 = -1 = 1 + 0 - 2 = S + M - G_{nm}$$

$$(f) V - E + F - R = 10 - 15 + 7 - 0 = 2 = 1 + 1 - 0 = S + M - G_{nm}$$

2.2.2 Solid Model Construction

Solid modeling methods are used to construct as well as store geometric data of solids in computers. There are six major categories of methods of constructing solid models: Instances or Parameterized Shapes, Cell Decomposition (including spatial occupancy enumeration), Sweep Representations, Constructive Solid Geometry (CSG), Boundary Representations (B-rep), and Wireframe Representations (Mortenson, 1997).

Of the six methods, CSG and B-rep are important and dominating the contemporary modeling systems. So we will briefly study these two methods.

Constructive Solid Geometry (CSG)

Constructive Solid Geometry (CSG) is a term for modeling methods that define complex solids as compositions of simpler solids (primitives). Boolean operators are used to execute the composition (Mortenson, 1997).

CSG representations of objects are ordered binary trees whose leaf or terminal nodes are either primitives or transformation data for rigid-body motions. The nonterminal nodes are either regularized Boolean operators (union, difference, and intersect) or rigid-body motions (translation and/or rotation) that operate on their two subnodes (or sub-solids). Each subtree of a node (not a transformation leaf) represents a solid resulting from the combination and transformation operations indicated below it. The root, of course, represents the final object.

In Figure 2.7, the leaf nodes at the bottom of the binary tree represent the primitives π_1 , π_2 and the translation Δx . The two internal nodes at the second row represent temporary results of the operations $(\pi_1 - \pi_2)$ and $\pi_2[\Delta x]$. The root node represents the final

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 19

Figure 2.7. CSG representation (adapted from Mortenson, 1997)

object.

Boundary Representation (B-rep)

The boundary of a solid separates points inside from points outside the solid. The boundary is the primary interface between the solid and the surrounding environment. *Boundary Representation (B-rep)* is a geometric modeling method which represents 3D objects in terms of the surfaces enclosing them.

Figure 2.8. B-rep Representation (adapted from Mäntylä, 1988)

Figure 2.8 illustrates the basic components of a B-rep model. In the figure, the surface of the object is divided into an enclosing set of faces (Figure 2.8a), each of which is represented in terms of its bounding polygon (Figure 2.8b), in turn represented in terms of edges and vertices (Figure 2.8c).

Various data structures have been developed for B-rep models (Mäntylä, 1988 and Boissonnat, 1984). Mäntylä (1988) introduced three types of data structures for B-rep in his book: polygon-based boundary models, vertex-based boundary models and edge-based boundary models.

Polygon-based Boundary Models This kind of models consists of polygonal faces. Because all edges of a polyhedron are straight-line segments, it is possible to shrink the boundary data structure considerably in this important special case.

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 21

In the simplest variation, faces are represented as polygons, each polygon consisting of a sequence of coordinate triples. A solid consists of a collection of faces grouped together in terms of, say, a table of face identifiers or a linked list of face nodes.

Vertex-based Boundary Models In a polygon-based boundary model vertex coordinates appear as often as the vertex appears in a face. This redundancy can be eliminated by introducing vertices as independent entities of the boundary data structure.

Faces of models are represented by consistently ordered vertex lists, namely the vertices in a list are ordered clockwise as seen from the outside of the model. The geometric coordinates of each vertex are stored in a vertex table so that vertices can be quoted by their indices and in turn reduce the needed storage space.

Edge-based Boundary Models An edge-based boundary model represents a face boundary in terms of a closing sequence of edges, or loop for short. Vertices of the face are represented only through edges. However, some useful information (e.g., face-face connectivity information) is not easily obtained by this data structure. A *winged-edge data structure* was introduced in order to address this problem. It goes one step further by representing also the loop information in edge nodes.

For simplicity and clarity, we will illustrate this kind of data structure using the cube shown in Figure 2.9.

Figure 2.9. A sample object for illustration of B-rep structures (adapted from Mäntylä, 1988)

A full winged-edge data structure for the model in Figure 2.9 is shown in Figure

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 22

2.10. Edges are identified by their start points ($vstart$) and end points ($vend$). fcw and $fccw$ are identifiers of the two faces which are connected along the corresponding edge. fcw is the face whose orientation coincides with that of the corresponding edge while $fccw$ is the face with counter-orientation against the edge. ncw and pcw denote the next and the previous edge in fcw , respectively. Similarly, $nccw$ and $pccw$ denote the next and the previous edge in $fccw$, respectively.

<i>edge</i>	<i>vstart</i>	<i>vend</i>	<i>fcw</i>	<i>fccw</i>	<i>ncw</i>	<i>pcw</i>	<i>nccw</i>	<i>pccw</i>
e_1	v_1	v_2	f_1	f_2	e_2	e_4	e_5	e_6
e_2	v_2	v_3	f_1	f_3	e_3	e_1	e_6	e_7
e_3	v_3	v_4	f_1	f_4	e_4	e_2	e_7	e_8
e_4	v_4	v_1	f_1	f_5	e_1	e_3	e_8	e_5
e_5	v_1	v_5	f_2	f_5	e_9	e_1	e_4	e_{12}
e_6	v_2	v_6	f_3	f_2	e_{10}	e_2	e_1	e_9
e_7	v_3	v_7	f_4	f_3	e_{11}	e_3	e_2	e_{10}
e_8	v_4	v_8	f_5	f_4	e_{12}	e_4	e_3	e_{11}
e_9	v_5	v_6	f_2	f_6	e_6	e_5	e_{12}	e_{10}
e_{10}	v_6	v_7	f_3	f_6	e_7	e_6	e_9	e_{11}
e_{11}	v_7	v_8	f_4	f_6	e_8	e_7	e_{10}	e_{12}
e_{12}	v_8	v_5	f_5	f_6	e_5	e_8	e_{11}	e_9

<i>vertex</i>	<i>first edge</i>	<i>coordinates</i>	<i>face</i>	<i>first edge</i>
v_1	e_1	$x_1y_1z_1$	f_1	e_1
v_2	e_2	$x_2y_2z_2$	f_2	e_9
v_3	e_3	$x_3y_3z_3$	f_3	e_6
v_4	e_4	$x_4y_4z_4$	f_4	e_7
v_5	e_9	$x_5y_5z_5$	f_5	e_{12}
v_6	e_{10}	$x_6y_6z_6$	f_6	e_9
v_7	e_{11}	$x_7y_7z_7$		
v_8	e_{12}	$x_8y_8z_8$		

Figure 2.10. The full winged-edge data structure (adapted from Mäntylä, 1988)

The data structure described so far assume that each face is simply connected, i.e. that it has just one boundary curve. However, sometimes it is necessary to include faces that have several boundary curves. An approach to address this problem is to add a sep-

arate loop node that models a simple boundary into the boundary model, and associate each face with a list of loops. References to faces would be replaced by references to the appropriate loops; for instance, face identifiers fcw and $fccw$ in edge nodes of the full winged-edge data structure would be replaced with the analogous loop identifiers lcw and $lccw$.

2.2.3 Graph Representation of 3D models

Figure 2.11. Examples of FAG representation

Ansaldi et al. (1985) presented a geometric modeling method by using a *Face Adjacency Graph (FAG)*, which is a relational graph structure based on a boundary representation of solid objects. In a FAG, nodes represent object faces, whereas edges and vertices are encoded into arcs and hyperarcs. Shpitalni & Lipson (2000) also presented a similar FAG method to represent the topological relationships among faces of a 3D model. Figure 2.11a is an L-shape object and Figure 2.11b is its corresponding FAG representation; Figure 2.11c is a non-manifold structure which is derived from Figure

2.11a and Figure 2.11d is the corresponding FAG of Figure 2.11c.

2.3 Graph Theory

Graph theory is a basic theory required in the unfolding methodology, so all necessary concepts and techniques in graph theory are discussed in the following sections.

2.3.1 Basic Concepts

In brief, a graph is a set of vertices that are connected by edges. More formally, a graph $G = (V, E)$ is a finite set of V and E , where V refer to the *vertices* and E refer to the *edges* (each of which connects two distinct vertices in V (Sahni, 1998)). Although the term ‘vertices’ and ‘edges’ are commonly used in graph theory, they will create confusion with the same terms ‘vertices’ and ‘edges’ as used in geometry. Therefore, the terms ‘nodes’ and ‘links’ will be used for the rest of this thesis in place of ‘vertices’ and ‘edges’ of a graph, respectively. And a graph G can in turn be represented as $G = (N, L)$. A link can be denoted by (i, j) , where i and j denote the two nodes that are connected by that link, respectively. Nodes i and j are *adjacent* if they are connected by a link. The *degree* d_i of a node i is the number of links connected to i .

Graphs are composed of nodes and links, as shown in Figure 2.12. Links with arrows in Figure 2.12b are *directed links*, while links with no arrows in Figure 2.12a are *undirected links*. Graphs are called *undirected graphs* (Figure 2.12a) if all links in them are undirected links. Similarly, graphs that have directed links are called *directed graphs* (Figure 2.12b). The methodology developed in this work is mainly based on undirected graphs, thus only undirected graphs are discussed here.

2.3.2 Graph Representation

In order to implement graph techniques in computers, a way to represent the mathematical structure of a graph as some kind of data structure is needed. There are three frequently used methods to represent graphs (Sahni, 1998): *adjacency matrix*, *packed-adjacency list*, and *linked-adjacency list*. Note that all methods discussed here are based

Figure 2.12. Graphs

on adjacency relationships between nodes within a graph. For simplicity, all examples of graph representation in this section are for the graph shown in Figure 2.12a.

Graph Representation Methods

Adjacency Matrix The adjacency matrix of a given graph $G = (V, E)$ with n nodes is an $n \times n$ matrix A with only elements 0 or 1. Provided that G is an undirected graph, the elements of A are defined as:

$$A(i, j) = \begin{cases} 1 & \text{if } (i, j) \in E \text{ or } (j, i) \in E \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

The adjacency matrix for Figure 2.12a is shown in Figure 2.13.

$$\begin{array}{c} \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Figure 2.13. Adjacency matrix

Packed-adjacency List Two one-dimensional arrays $h[0 : n]$ and $l[0 : x]$ are used in

Figure 2.14. Packed-adjacency list

the definition of the packed-adjacency list of a graph G ($G = (V, E)$, $|V| = n$, $|E| = e$). $x = 2e - 1$ if G is an undirected graph. Firstly, add all nodes connected with node 1 to l and then add those connected with node 2, 3 and so on. Secondly, to construct h , let it satisfy that when $h[i - 1] < h[i]$, positions of all nodes connected with i be $l[h[i - 1]]$, $l[h[i - 1] + 1]$, $\dots$, $l[h[i] - 1]$; when $h[i - 1] \geq h[i]$, no node is connected with i . $l[h[i - 1]]$, $l[h[i - 1] + 1]$, $\dots$, $l[h[i] - 1]$ are collectively called the packed-adjacency list of node i . Orders of nodes in packed-adjacency lists can be arbitrary. Figure 2.14 shows the packed-adjacency list for Figure 2.12a.

Linked-adjacency List Linked-adjacency lists are stored as chains. The i th row of a list is a chain indicating the connectivity between node i and other nodes. The data of each node in the i th chain denotes a node which is connected with node i . The linked-adjacency list for Figure 2.12a is shown in Figure 2.15.

Time Complexity

Table 2.3 is the time complexity analyses of the main operations of these three graph representation methods (Sahni, 1998).

From Table 2.3, we can see that the adjacency matrix has the best time complexity. However, it costs more in terms of storage space than the other two methods. Linked-adjacency list seems to have the best balance between time complexity and space complexity.

2.3.3 Graph Traversal

There are two commonly-used graph traversal methods: *Breadth First Search (BFS)* and *Depth First Search (DFS)*. Both of them are popular and have the same time complexity,

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 27

Figure 2.15. Linked-adjacency list**Table 2.3.** Time complexity analysis

<i>Operations</i>	<i>Adjacency Matrix</i>	<i>Packed-adjacency List</i>	<i>Linked-adjacency List</i>
Determine the existence of a link	$O(1)$	$O(\log n)$	$O(\log n)$
Get link number	$O(n^2)$	$O(1)$	$O(n^2)$
Get node number	$O(1)$	$O(1)$	$O(1)$
Add a link	$O(1)$	$O(n + e)$	$O(1)$
Delete a link	$O(1)$	$O(n + e)$	$O(1)$
Get the degree of a node	$O(n)$	$O(1)$	$O(\log n)$

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 28

but DFS has an empirically higher efficiency and thus is used more often (Sahni, 1998). Let us use the graph shown in Figure 2.12a to illustrate these two methods.

Breadth First Search (BFS) Assuming that a BFS is started from node 1, the first step is to get the node set nearest to node 1, which is obviously $\{2, 3, 4\}$. Secondly, identify the nearest node set to the first element of this set, namely, node 2. What we get is a one-element set $\{5\}$. The nearest node to 3 is 5 as well; however, because node 5 has already been traversed, we proceed to node 4, to which the nearest set is $\{6, 7\}$. Finally, node 8 is traversed from 5. Figure 2.16a shows the traversal order of BFS of Figure 2.12a.

Figure 2.16. (a) BFS; (b) DFS

Depth First Search (DFS) Starting from a node v (e.g. node 1), DFS follows the following steps: firstly, label v as a traversed node, and then select an untraversed node u (e.g. node 2) which is adjacent to v . Stop searching if there does not exist such a node u ; otherwise, start a new DFS from u . Select another untraversed node which is adjacent to v when the DFS procedure started from u has finished, and start another DFS from this node until all nodes are traversed. Figure 2.16b shows the traversal order of DFS of Figure 2.12a.

2.3.4 Spanning Trees

Given an undirected graph $G = (V, E)$, provided that all nodes are connected by links, a subgraph $G' = (V, E')$ is called a spanning tree of G when each link $e \in E'$ also satisfies that $e \in E$ and there do not exist any cycles in G' nor any separated components.

The number of possible spanning trees of an undirected graph with no self-loops is equal to any of the minors of its degree matrix resulting from the erasure of a row and a corresponding column (Even, 1979).

The degree matrix of a graph is defined as follows:

$$D_{ij} = \begin{cases} d(i) & \text{if } i = j \\ -k & \text{if } i \neq j \end{cases} \quad (2.8)$$

where $d(i)$ is the degree of the i^{th} node, i.e. the number of links that are incident on this node. k is the number of links between node i and j .

Figure 2.17. Graph with 4 nodes and 5 links

For illustration purposes, consider the graph G shown in Figure 2.17. This graph has 4 nodes and 5 edges. According to the definition, the degree matrix D of graph G is given by

$$D = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} \quad (2.9)$$

The required minors of D are obtained by computing the determinant of any sub-matrix of D resulting from the deletion of one row and the corresponding column from D . For example, by deleting the first row and the first column, the following minor is obtained:

$$\begin{vmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{vmatrix} = 8 \quad (2.10)$$

Hence the total number of spanning trees of G is eight. All the spanning trees of G are listed in Figure 2.18.

Figure 2.18. Spanning trees

2.3.5 Spanning Tree Enumeration

Many studies have been conducted to enumerate all spanning trees of directed or undirected graphs. Various algorithms have been developed for this purpose, and can be classified into two types. The first type, most developed before 1990s, used a method named "backtracking" (Read & Tarjan, 1975; Gabow & Myers, 1978). Gabow & Myers (1978) proposed an algorithm using the backtracking method, which uses $O(NV + V + E)$ time and $O(V + E)$ space, where N , V and E are the number of spanning trees, nodes and links, respectively.

Note:

T is a spanning tree of G , e_i is the index of the i^{th} link, $f=e_1$ and $g=e_4$

Figure 2.19. Deriving a new spanning tree from a current tree T

In the last decade or so, several algorithms using a compact output technique have been proposed by Kapoor & Ramesh (1992, 1995, 2000), Matsui (1993, 1997) and Shioura et al. (1997). The basic strategy behind this technique is to find a new spanning tree by exchanging one edge of a current spanning tree. Consider the same graph G from Figure 2.17 and shown here again with the links labeled in Figure 2.19a, together with one possible spanning tree T in Figure 2.19b. For any link $f \in T$, the deletion of f

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 31

from T yields a graph (denoted by $T \setminus f$) with two and only two separated components. For example, if f is the link e_1 then $T \setminus f$ is as shown in Figure 2.19c. The fundamental cut associated with T and f is defined as the set of links from the original graph G that can connect these two components, and it is denoted by $Cut(T \setminus f)$. For the illustrated example in Figure 2.19, this set is given by:

$$Cut(T \setminus f) = \{e_1, e_4, e_5\}$$

Likewise, the fundamental cycle associated with T and another link $g \notin T$ is the set of links contained in the unique cycle $T \cup g$ and is denoted by $Cyc(T \cup g)$. If g is the link e_4 , then $T \cup g$ is as shown in Figure 2.19d and the corresponding set in the fundamental cycle is given by:

$$Cyc(T \cup g) = \{e_1, e_2, e_4\}$$

By definition, $T \setminus f \cup g$ is a spanning tree for any $f \in T$ and any $g \in Cut(T \setminus f)$, and such a tree is shown in Figure 2.19e based on the current example where f is the link e_1 and g is e_4 . Similarly, $T \cup g \setminus f$ is also a spanning tree for any $g \notin T$ and any $f \in Cyc(T \cup g)$, and it leads to the same tree as before, as shown in Figure 2.19e.

The above idea alone is not sufficient to enumerate all spanning trees systematically. This is because if $T_1 = T_2 \setminus f \cup g$, then $T_2 = T_1 \setminus g \cup f$ also holds and therefore exactly the same spanning trees (T_1 and T_2) will be traversed again and again and the enumeration will never end. To break this infinitude, all links of the graph are indexed into a link set $\{l_1, l_2, \dots, l_L\}$ (where L is the number of links in the graph) in such an order that $\{l_1, l_2, \dots, l_{N-1}\}$ (where N is the number of nodes in the graph) is a spanning tree T^0 of the graph. The total order over the links is defined by their indices as $l_1 < l_2 < \dots < l_L$. For any spanning tree T' , $\phi(T')$ denotes the spanning tree $(T' \setminus f \cup g)$, where f is the link with largest index in T' and g is the link with smallest index in $Cut(T' \setminus f)$. The spanning tree T' is defined as a child of $\phi(T')$ and $\phi(T')$ is the parent of T' . A tree structure is then constructed, denoted by (Γ, ϕ) , with the node set Γ and link set $\{(T, T') \in \Gamma \times \Gamma \mid T' \neq T^0, T = \phi(T')\}$. The spanning tree T^0 is the root of this tree structure. The compact output method searches this tree structure by using a tree search rule and outputs all the

CHAPTER 2. GEOMETRIC MODELING AND GRAPH THEORETIC PRELIMINARIES 32

spanning trees of the given graph.

Shioura et al. (1997) proposed an algorithm which can finish spanning tree enumeration within $O(N + V + E)$ time and with $O(V + E)$ space. So far it is still believed optimal in terms of time and space complexity. Matsui (1997) presented another algorithm with a time complexity of $O(NV + V + E)$. Although it is less efficient than Shioura et al. (1997), it has a unique merit. The algorithm itself follows a special rooted tree structure on the skeleton graph of the spanning tree polytope. The rule by which the rooted tree structure is traversed is irrelevant to the time complexity. In this sense, the algorithm is flexible. If a depth-first rule is employed, the memory requirement can be reduced to $O(V + E)$. A breadth-first implementation requires as much as $O(NV + E)$ space, but when a parallel computer is available, this might have an advantage. When a given graph is weighted, the best-first search rule provides a ranking algorithm for the minimum spanning tree problem. The ranking algorithm requires $O(NV + V + E)$ time and $O(NV + E)$ space when a minimum spanning tree is obtained.

CHAPTER 3

Literature Review

The use of paperboard as packaging buffers is becoming an attractive and viable alternative in the packaging industry, but very little research on the design of such buffers has been carried out. However, some research in the areas of origami and sheet metal design is relevant to this field of packaging.

3.1 Study of Origami

Origami is the Japanese name of the old folk art of folding paper into representations of insects, birds, animals, human figures, inanimate objects, and abstract shapes. In the purest form of origami, the figure is folded from a single uncut square of paper. In the past 50 years, the complexity of origami figures has increased dramatically. Some research work in this field are confined to flat origami, i.e. sheets of paper are folded into objects which are flat or piecewise flat structures (Agarwal et al., 1997, Bern & Hayes, 1996, Demaine et al., 2000 and Hull, 1994, 1996, 1997). Lang (1996) proposed an algorithm developed for generating the crease pattern for folding into various origami

designs. However, the generated crease pattern can only be folded into flat figures which do not contain some of the complex features that make up 3D folded structures. Some other topics in origami are related to polyhedra and polytopes (Agarwal et al., 1997, Bangay, 2000, Bern et al., 1999, Biedl et al., 1998, Buekenhout & Parker, 1998, Demaine et al., 2000, Demaine et al., 2002, Demaine et al., 1999, Lubiw & O'Rourke, 1996, Maehara, 2000, O'Rourke, 1998 and Schlichenrieder, 1997), which are more related to 3D folded structures. Thus, we are most interested in these topics.

Glossaries of origami are derived from algebraic geometry, which is the study of geometries derived from algebra. Some definitions in algebraic geometry are slightly different from that in geometry. A polyhedron in origami is defined as the intersection of a finite number of half-spaces. If the resulting polyhedron is bounded, it is called a polytope (Schlichenrieder, 1997). Thus, a polytope in origami is a polyhedron in geometry. Note that in this section, the meaning of polyhedron follows that from geometry.

3.1.1 Unfolding Types

According to the position of seams, unfoldings can be classified into two major categories: *edge unfoldings* and *general unfoldings*. *Edge unfoldings* refer to those unfoldings that contain only seams which are the common edges of the 3D folded structure, while *general unfoldings* are defined as those unfoldings that contain at least one non-common edge seam. In packaging and many other practical applications, edge unfoldings are preferred in most cases, for it maintains the strength and integrity of the structures. However, as a matter of fact, many structures do not have an edge unfolding at all. Figure 3.1(a) shows such a structure. It is a cube with a depression in it. Faces of the depression can only be flattened inside the ring of the top face according to their topology. However, the total area of all the faces of the depression is greater than that within the ring, which means the ring cannot hold all the faces of the depression; consequently, this structure does not have any non-overlapping edge unfolding. However, if we add two vertex-vertex non-common edge seams to the top face such as S_1 , and S_2 in Figure 3.1(b), it can be flattened to a general unfolding with no overlapping. Moreover,

the top face can be treated as having been split into two faces, therefore $S1$ and $S2$ can be looked upon as common edges between these newly created faces and in turn, general unfoldings are actually edge unfoldings of the modified structure. This change of thinking in unfolding types is significant, for it may provide us an approach to deal with different types of structures using a uniform methodology. If we have a methodology to generate edge unfoldings for a given unfoldable structure, (i.e. a structure that can be unfolded into an edge unfolding), we can use this methodology to deal with those unfoldable structures as well, by modifying them into unfoldable structures without changing their structure or shape.

Figure 3.1. An example of unfoldable structures

3.1.2 Unfoldability

The study of unfolding types has generated an interest in the unfoldability of structures. Actually, unfoldability of structures is one of the main concerns in origami. An open question on folding/unfolding (Fukuda, 1997 and Schlichenrieder, 1997) is: can every convex polytope be cut along edges and unfolded flat to a single non-overlapping piece? As mentioned in the previous section, many structures do not have non-overlapping edge unfoldings. However, these structures are all non-convex polytopes. Nobody has declared having found a convex polytope with all its possible unfoldings overlapping. Although it is widely conjectured that every convex polytope contains at least one non-overlapping edge unfolding, it still remains an open question.

However, statistical data can throw up some clues in finding out the law. Figure 3.2

Figure 3.2. Probabilities of overlapped unfoldings of randomly generated polytopes (adapted from O'Rourke, 1998)

shows the probability relationship between the number of polytope vertices and its overlapped unfoldings. It can be conjectured from Figure 3.2 that as $n \rightarrow \infty$, the percentage of the overlapping edge unfoldings of randomly generated polytopes of n vertices approaches 100%. When $n \geq 70$, 99% of the edge unfoldings of the polytope lead to overlap. However, no one has found a specific polytope with exactly 100% of its edge unfoldings overlap, nor has anyone proven that there exists at least one non-overlapping edge unfolding for each and every convex polytope.

It should be noted that the above statistical study is confined to convex polytopes. Bern et al. (1999) has constructed several ununfoldable non-convex polytopes that do not have any non-overlapping edge unfolding. The unfoldabilities and unfolding methodologies of non-convex polytope, and furthermore, non-manifold structures, are a relatively unstudied field in origami. However, our main interests lie in them as well. Though the unfoldabilities of non-convex polytopes and non-manifold structures are unknown, it does not mean that we cannot flatten them into non-overlapping flat layouts. On the contrary, the goal in this thesis is to develop a methodology to flatten such structures, mainly non-convex polytopes and non-manifold structures.

3.1.3 Unfolding Methodologies in Origami

Although the unfolding methodologies for complex structures, i.e. non-convex simple polyhedra, non-simple polyhedra and non-manifold structures, still remain relatively unstudied, various methodologies for convex polytopes are well developed.

If the cuts are permitted to be arbitrary segments on the convex polytopes, rather than only edges, then the non-overlapping unfoldings are guaranteed at least in two ways: one is called *star unfolding*, the other is called *source unfolding*. Given a convex polytope P , fix a source point $x \in P$, the star unfolding with respect to x cuts P along n shortest paths from x to each of the n vertices on P (Agarwal et al., 1997). A shortest path is the minimal distance along the surface of the polytope from x to the i^{th} vertex (as shown by a pyramid structure example in Figure 3.3). The result is a non-overlapping unfolding of the polytope P .

A second unfolding, more difficult to define but easier to prove non-overlapping, is the *source unfolding*, which has been studied since the turn of the last century. Given two points x, y on P , $y \in P$ is a *ridge point* with respect to x if there is more than one shortest path between x and y . The set of all ridge points with respect to x form a *ridge tree* T_x embedded on P , whose leaves are corners of P , and whose internal vertices have degree at least three and correspond to points of P with three or more distinct shortest paths to x .

If we cut P along the ridge tree T_x and embed the resulting set in R^2 , *source unfolding* of P is obtained. Figure 3.3 shows the ridge tree for a pyramid structure.

The thick solid lines are common edges between faces. Thin solid lines are shortest paths to the five vertices from the source x , i.e. they are the cut set of star unfolding. Dashed lines are the edges of the ridge tree, i.e. they are the cut set of source unfolding.

Figure 3.5 and 3.4 are examples of star unfolding and source unfolding for the pyramid shown in Figure 3.3, respectively.

Several algorithms for developing non-overlapping edge unfoldings are introduced for convex polytopes. However, these algorithms are also suitable for generating edge

Figure 3.3. Pyramid, front (a) and side (b) views. Shortest paths to five vertices from source x are shown as thin solid lines; the ridge tree is shown as dash lines. (adapted from Agarwal et al., 1997)

Figure 3.4. Star unfolding for the pyramid (adapted from Agarwal et al., 1997)

Figure 3.5. Source unfolding for the pyramid (adapted from Agarwal et al., 1997)

unfoldings for more complex structures that contain at least one non-overlapping edge unfolding.

Schlichenrieder (1997) presented a backtracking algorithm in his Ph.D. thesis. This algorithm begins with choosing a root face f from the structure and adding it to the layout N . Note that from here onwards the meaning of f is changed from root face to last added face of N . Then choose a face g from among the neighbors of f in the remaining structure and flatten it to the same plane as f . If the flattened face g does not overlap with any face in N , then add it to N , otherwise, return it to the remaining structure and choose another neighbor g' of f and follow the above steps. If all the neighbors of f have been tested, then return f to the remaining structure and designate the last added face of N as f and repeat testing until all the faces in the remaining structure have been tested without success, then announce failure.

Based also on a backtrack method, Bangay (2000) developed two algorithms called *the object space algorithm* and *the image space algorithm*. The main idea of *the object space algorithm* is to adjust priorities of the faces so as to reach the goal faster. Each

time a face is found overlapping with one of the faces in the existing layout, it is returned to the remaining structure after its priority has been adjusted according to some criteria. And then another face with a better priority is chosen from the remaining structure to be tested. At the end of each attempt to add a face to the existing unfolding, the priority values are raised according to overlapping difficulties that it encounters, e.g. how many times a face is found at the root of the two branches which contain the two colliding faces; how many times a face is involved in a collision, when the face is already a part of the unfolding; and how many times a face collides when it is in the process of being added.

The image space algorithm manipulates a partially correct folded layout directly in an attempt to improve the level of correctness. The action of this algorithm can be compared to unraveling only the tangled portion of the layout. When a flattened face overlaps with some other faces in the layout and furthermore, it cannot be attached to any other locations in the current layout without overlapping, then add it to the layout anyway and delete those other faces that caused the overlapping, and repeat testing until all the faces have been tested.

Compared with the image space algorithm, the object space algorithm seems to have a much higher efficiency.

The algorithms discussed are used for generating single unfoldings, i.e. they will terminate when an unfolding is found. However, there may be a desire to enumerate all possible non-overlapping unfoldings. Although he did not develop such a methodology, Schlichenrieder (1997) had suggested how such a goal can be achieved. Note that the topology of a structure can be represented by a graph, and the topologies of all edge unfoldings of it are spanning trees of this graph. Accordingly, we may get all these unfoldings by enumerating all spanning trees of the graph, and then check for overlapping.

3.2 Study of Sheet Metal Design and Thin-walled Objects

A large number of manufactured goods are produced by folding thin sheets of material into 3D shapes. File cabinets, electronic component cases, automobile body and container manufacturers fold metal or corrugated sheets into appropriate 3D shapes. Traditional design methods often require highly skilled pattern makers to develop 2D patterns and moreover, the process of laying out the models of 3D shapes is time consuming, prone to error and potentially wasteful of the raw material. Thus, research work had been carried out on the design of sheet metal products (Bush & Sëquin, 1999, Fu, 1993 and Radin et al., 1997).

3.2.1 Unfolding Methodologies in Sheet Metal Design and Thin-Walled Objects

See Toh et al. (1995) developed a feature-based flat pattern development system for sheet metal parts. Their flat patterning of a 3D design involves checking of bend allowances resulting from metal stretching at bends, as well as unfolding of walls and mapping of features such as slots, cutouts, etc. This system works pretty well on some sheet metal parts. However, some other features cannot be handled, e.g. weld lines that must be split in the flattened pattern. Furthermore, although feature recognition and classification are extensively studied (Chuang & Henderson, 1994, Fu et al., 2003, Gadh & Prinz, 1995, Holland et al., 2002, Jagirdar et al., 2001, Ji & Marefat, 2003, Li & Liu, 2002, Shah & Mäntylä, 1995 and Zhu & Menq, 2002), no standard feature classification give a unique representation to a given 3D object (since the object can be decomposed into different features according to different points of view). Therefore, this ambiguity will make the problem more difficult when using feature based methods. Thus, their methodology of unfolding will meet with difficulty when faced with more complex objects.

Shpitalni & Lipson (2000) proposed an algorithm for determining the flat pattern of sheet metal products. This algorithm is based on a heuristic search on the connectivity graph, which represents the topological connectivity of a sheet metal product. Nodes of

the graph correspond to faces and links correspond to connections between faces. Figure 3.6 is an example of a product and its corresponding graph.

Figure 3.6. (a) An open box; (b) its graph representation. (adapted from Shpitalni & Lipson, 2000)

In principle, any flat pattern is a spanning tree of the graph. Depending on the optimality criteria, weights can be assigned to the links corresponding to their desirability as bends. Determination of the flat pattern is reduced to finding a maximum weight spanning tree. However, in contrast with abstract spanning trees, the flat pattern must comply with certain constraints, e.g. faces in the unbent flat pattern must not overlap. This restriction introduces the possibility that a compliant spanning tree may not exist at all.

The implementation of their solution algorithm is based on the A* approach, a well-established artificial intelligence (AI) technique for optimal heuristic search on a graph. The bend assignment problem is formulated as a graph with nodes and links. The nodes of the graph correspond to states in the problem domain, each associated with a valid flat pattern set represented by a binary vector with a digit set to one for active links. Initially (root node), all links are detached, and so the state vector is zero. This corresponds to a flat pattern set composed of the individual faces, each as a detached unit, which is always a valid solution. The graph links correspond to transitions from state-to-state, done by joining two flat pattern components in a set while ensuring no overlap is introduced, no product edge is used more than once and no loop is closed. This ensures that all valid multi-component layouts are considered on the ‘path’ to evaluate optimal single component flat patterns.

In order to define the search goal, a measure of state cost is necessary. In the A* algorithm, the cost associated with each state is composed of two terms: $g(n)$, corresponding to the actual cost required to reach the current state from the initial state, and $h(n)$, corresponding to an optimistic estimation of the remaining cost required to reach a goal state from the current state, where n is the current state. The cost functions are dependent on the overall goal of the unbending procedure. In a simple case, the cost $g(n)$ is a sum of the costs c_b of individual bend lines used so far, plus a cost c_d associated with the number of detached components. Typically, selecting the long bend lines gives better layouts, so cost is assigned in inverse proportion to length. Since it is usually simpler to manufacture a product with fewer components, it is desirable that $c_d \gg c_b$, so that bends will be preferred over welds where possible. The remaining cost $h(n)$ can be estimated optimistically by assuming that at best, for k faces with m links already selected, the best $k - m - 1$ of the remaining links will be used. A bound on the number of components can be obtained from topological consideration.

The above unfolding methodology can yield good solutions, including products requiring multiple components. However, cost functions may be different (i.e. different optimality criteria are applied) for different applications.

Aiming at designing flat patterns for sheet metal products, Wang (1997) (see also Wang & Bourne, 1996) proposed an unfolding strategy named develop-first- decompose-later strategy. 3D objects are firstly flattened into a 2D flat pattern. Based on this 2D pattern, a set of manufacturing feasibility checkers (e.g. part unfoldability, tool accessibility and product assemblability) is then used to evaluate whether it is manufacturable or not. The product is recursively decomposed if it fails to pass these tests. The desired decomposition is the one with fewer parts. A* search algorithm is used to search for the desired decomposition. Since the decomposition search is performed on a 2D flat pattern (the topology of which is a tree), the complexity of the method is linear. However, this method does not consider various optimality criteria and even for geometric compactness, it cannot guarantee to get the optimal result (since only one 2D flat pattern is generated and used for decomposing, which may not be optimal). Moreover, for

some 3D objects that contain non-overlapping edge unfoldings, multi-component results may be generated (this may happen when the first generated 2D flat pattern contains overlapping).

Lin & Yang (1994, 1998) used a different way to achieve the same goal. They put their emphasis on the choice and assignment of seams, which can be classified into four styles, as shown in Figure 3.7: (a) seams coinciding with common edges are *common edge seams*. Other seams are not located on common edges, thus they are *non-common edge seams*. Further more, in the scope of non-common edge seams, there are three specific types, namely (b) *vertex-to-edge seams*, (c) *edge-to-edge seams* and (d) *vertex-to-vertex non-common edge seams*.

Based on a mechanism theory, they put forward the following seam line equation that indicates the number of seams that are needed for the purpose of unfolding a structure

$$s = e - f + 1 \quad (3.1)$$

where s denotes the number of seams needed, e denotes the number of common edges and f denotes the number of faces.

Figure 3.7. Seam line styles (adapted from Lin & Yang, 1998)

In Lin & Yang (1998), only common edge seams are used when generating unfoldings and since common edges connect two vertices, the seams can be assigned to these vertices. Vertices located on free edges are called ‘*boundary vertices*’, while the others

are called ‘*inner vertices*’. The surrounding angle of a point on a flat plane is 2π . If the sum of surrounding angles of all faces at an inner vertex is less than 2π , then at least one seam is needed at this vertex. If the sum of surrounding angles is greater than 2π yet less than 4π , then at least two seams are needed to split the corner into two or more portions at that vertex. The minimum required number of seams at a vertex would increase by 1 for every increase of 2π in the sum of surrounding angles.

In summary, the minimum number of required seams at an inner vertex can be calculated from the following formula:

$$2\pi(n_1 - 1) < \sum_{i=1}^m \theta_i \leq 2\pi n_1 \quad (3.2)$$

(except for 2π vertices, where $n_1 = 0$) where

n_1 = minimum number of seams at an inner vertex

m = the number of faces around a vertex

$\theta_i = i^{th}$ surrounding angle around a vertex

For a boundary vertex, the Equation (3.2) should be modified because the free edge itself can be looked at as a seam. Lin & Yang (1998) presented the new equation for a boundary vertex as:

$$2\pi n_1 < \sum_{i=1}^m \theta_i \leq 2\pi(n_1 + 1) \quad (3.3)$$

When the structure contains non-straight common edges, they must be seams. This is because non-straight lines cannot be used as fold lines. Therefore, if a vertex lies on n_2 non-straight common edges, it needs at least n_2 seams.

In conclusion, the minimum number of required seams at vertex i , say s_i , is:

$$s_i = \max(n_1, n_2) \quad (3.4)$$

After the required number of seams for the structure is identified as well as the minimum number for each vertex, seams can be subsequently assigned to vertices based on the above information. Lin & Yang (1998) enumerated all possibilities by assigning the required number of seams to all vertices sequentially. But apart from all possibilities, the

enumeration may also yield illegitimate results due to the following two possible violations: (1) the arrangement of seams caused the structure to be separated into at least two parts, (2) different faces of the generated flat layout partially overlap. Violation of item 1 can be checked by identifying whether the topological connectivity of the flat layout is a spanning tree of the corresponding graph representation of the corresponding 3D structure. Violation of item 2 cannot be checked until the flat layout is generated.

Once the seams are assigned to different common edges, flat layout generation procedure should be launched. In Lin & Yang (1998), faces of the structure are firstly mapped to the XY plane. Then, each time when generating a flat layout, mapped faces are linked together according to their original connectivity. This procedure is to avoid the repetition in rotating and translating the faces.

Chuang & Huang (1996) proposed an unfolding strategy by using a feature decomposition method. 3D models are decomposed into a collection of depression or protrusion features. The purpose of feature extraction is to decompose the surface wrapping up a closed space, i.e. the solid model, into a number of developable parts without overlapping. Each decomposed feature is represented as a face-edge graph. The flattening of a feature is based on its corresponding face-edge graph. A base face is first selected from the flattening feature. Thereafter, BFS and DFS spanning trees will be constructed and all other faces will be rotated to the plan of the base face around their folding axes, according to the connectivity indicated by the spanning trees. Since the flattened 2D model contains multiple components, these components will be at last interactively nested to a plane.

All of the above-mentioned methods (i.e. Shpitalni & Lipson, 2000, Lin & Yang, 1998, Wang, 1997 and Chuang & Huang, 1996) have their respective objectives and ways to unfold a given 3D structure. As compared to these existing approaches, this current work tries to deal with additional issues related to unfolding. Firstly, we tried to retrieve all non-overlapping edge unfoldings for a 3D structure for later study and tried to obtain different optimal results based on different optimality criteria. For those structures containing depressions, protrusions and through holes (for which non-overlapping edge

unfoldings may not exist), we give advice on how to modify the original structure to make it possible to be unfolded. Secondly, we tried to deal with structures containing complex features that may pose problems to the existing unfolding methods (e.g. the situation where more than two faces are sharing one single edge). Lastly, we also tried to improve our unfolding methodology efficiency by using topological recognition and make it suitable for distributed computing.

3.3 Other Related Work

There are some other research work that bear some relationship with the folding/packaging applications and graph-theoretic algorithms developed in this thesis. For example, Dai & Jones (1999) treated and analyzed some foldable structures as mechanisms that alter their structure when erected or folded. Dai & Jones (2002) used a screw representation to describe the mobility of a carton and its manipulation during the erection and folding processes. The representation can describe the manipulation in a quantitative way in order to provide information for manipulation planning in packaging automation. A similar study has been carried out in Liu & Dai (2002) where an origami-type carton is translated into an equivalent mechanism system. The creases of the carton are analogous to joints in the mechanism while the carton panels are viewed as links. The folding sequence can then be produced by a mechanism analysis. Lu & Akella (1999) also introduced a motion-planning algorithm that generates all folding sequences for a carton by modeling it as a many degree of freedom robot manipulator with revolute joints and branching links.

Another interesting work is that of Kling et al. (2002) where geometric theories have been developed for generating double-periodic folding patterns within one sheet. The resulting structures may be considered as sheets with discrete raised patterns which give the sheet desirable properties suited for a variety of purposes and yet is economical to produce since the manufacturing process is essentially folding only.

In her thesis (Sun, 1999), Sun has studied the possibilities of folding orthogonal poly-

gons into orthogonal polyhedra, with or without given folding directions (i.e. mountain and valley folds). She proved that when directions are given, the problem is polynomial complete, while if the directions are not given, the problem is NP-complete. Another interesting issue in this thesis is that she has found some example polygons that cannot be folded into a polyhedron if folds are only to be executed one at a time, or if the material is rigid.

Another related work is the design of pop-ups. Chatani (1984, 1986) has presented many exquisite pop-up patterns for origamic architectures and greeting cards in his well-known books. Also, Lee et al. (1996) studied the geometric modeling and simulation of pop-up books. However, the mathematical formulations in this work focus mainly on shape and motion information of the pop-up mechanisms and not on the topology.

Roth & Wybenga (2000) has built up a resource database of some 550 ready-to-use patterns and structural designs for folding cartons, trays, tubes, sleeves, and wraps. For packaging professionals, this book provides them the ability to obtain the flat patterns for various generic container box designs. However, packaging cushions/buffers made of (corrugated) paperboard are still relatively new and so there are as yet no well-established standard flat patterns for their manufacture.

The application of graph-theoretic methods to the topological design of structural systems has been investigated by Yetis & Saitou (2002), Lyu & Saitou (2003), Lee & Saitou (2003) and Cetin & Saitou (2004a,b). Aimed at helping product designers to obtain optimal designs (according to different optimality criteria like maximum structural strength, maximum assemblability, modularity and in-process adjustability) prior to the detailed component design phase, the methods described in these works represent the structural or product geometry by a graph of its topology. The graph is then decomposed into several disconnected subgraphs, each subgraph representing a separated component. The goal of the graph decomposition is defined by an objective function according to specific optimality criteria. And since there are many ways to decompose a graph, genetic algorithms are used as the basic search strategy to find the optimal decomposition and hence the optimal division of a product into smaller separated components.

CHAPTER 4

Enumeration of Flat Layouts

Based on Schlichenrieder (1997) and Shpitalni & Lipson (2000), the Face Adjacency Graph (FAG) of a valid flat layout is a spanning tree of the FAG of the corresponding 3D structure. An unfolding methodology founded on a spanning tree enumeration method was proposed and implemented in this chapter. As long as the 3D structure has been input into the computer using the appropriate geometric modeling scheme (also described in this chapter), the methodology can exhaustively enumerate all possible non-overlapping flat layouts.

4.1 Modeling of the structures

Both B-rep and Graph Representation methods will be used to model the 3D structures. By using B-rep, geometric attributes of structures can be obtained so as to get the accurate positions and dimensions of these structures. However, B-rep is not efficient when topological information is to be derived. The problem can be addressed by using graph representations, which can provide topological relationships between faces of structures.

4.1.1 Geometric Modeling

The winged-edge structure representation described in Chapter 2 is not applicable to non-manifold structures for it constrains one edge to be shared by two and only two faces. In non-manifold structures, it is common that an edge is shared by only one (namely a free edge) or more than two faces. Thus a modified B-rep data structure is to be used to describe both manifold and non-manifold structures.

Figure 4.1. Illustration of applied B-rep data structure

Figure 4.1 illustrates the geometric modeling method that will be used in this work. The various entities of the data structure are described as follows:

- Vertex** A vertex represents a point in 3D space, and it can be defined by its coordinates in the x , y , z directions.
- Edge** An edge is a straight-line segment between two vertices.
- Edgeuse** Edgeuse is the oriented use of an edge. It defines the orientation of the edge.
- Loop** A loop is a connected component of the boundary of a face. It is defined by a collection of edges. The direction of loop is indicated by the directions of edges, which means the orientations of edgeuses should be consistent.

- Loopuse** Loopuse is the oriented use of a loop. The direction of loopuse can either coincide with the direction of the corresponding loop or be reversed.
- Face** A face is an open finite connected subset of a plane surface, whose boundary is defined by a set of loops. Inner loops of a face are called *rings*. The direction of a face can be obtained by the directions of loopuses, using the right-hand rule.
- Faceuse** Faceuse is the oriented use of a face. A model can use either or even both sides of a face.
- Model** A model is a collection of faces, loops, edges and vertices. In this thesis, a model refers to a 3D folded structure to be flattened.

A simple example is shown in Figure 4.2 to give a full illustration of this data structure. This is a non-manifold structure that contains three faces (f_1, f_2, f_3) sharing one common edge (e_3). The full data structure for this example is given in Figure 4.3.

Figure 4.2. An example to show data structure

4.1.2 Topological Modeling

The topological relationships between the faces of a given 3D folded structure are represented by means of the face adjacency graph described in Chapter 2. However, the FAG model has two problems that have to be examined before using it as the topological information provider.

<i>vertex</i>	<i>edges</i>	<i>coordinates</i>			
v_1	e_1, e_4	$x_1y_1z_1$			
v_2	e_1, e_2	$x_1y_1z_1$			
v_3	e_2, e_3, e_7, e_{10}	$x_1y_1z_1$			
v_4	e_3, e_4, e_5, e_8	$x_1y_1z_1$			
v_5	e_5, e_6	$x_1y_1z_1$			
v_6	e_6, e_7	$x_1y_1z_1$			
v_7	e_8, e_9	$x_1y_1z_1$			
v_8	e_9, e_{10}	$x_1y_1z_1$			

<i>edge</i>	<i>vstart</i>	<i>vend</i>	<i>loops belong to</i>			
e_1	v_1	v_2	l_1			
e_2	v_2	v_3	l_1			
e_3	v_3	v_4	l_1, l_2, l_3			
e_4	v_4	v_1	l_1			
e_5	v_4	v_5	l_2			
e_6	v_5	v_6	l_2			
e_7	v_6	v_3	l_2			
e_8	v_4	v_7	l_3			
e_9	v_7	v_8	l_3			
e_{10}	v_8	v_3	l_3			

<i>loop</i>	<i>face</i>	<i>edgeuse</i>
l_1	f_1	e_3 , positive
		e_4 , positive
		e_1 , positive
		e_2 , positive
l_2	f_2	e_3 , positive
		e_5 , positive
		e_6 , positive
		e_7 , positive
l_3	f_3	e_3 , positive
		e_8 , positive
		e_9 , positive
		e_{10} , positive

<i>face</i>	<i>loopuse</i>
f_1	l_1 , positive
f_2	l_2 , positive
f_3	l_3 , positive

Figure 4.3. Full data structure for example in Figure 4.2

Problem I: Different Structures Represented by the Same FAG

A FAG alone cannot fully define the topology of a non-manifold 3D structure. As an example, Figure 4.4a, b and c are three different non-manifold structures and Figure 4.4c even contains a *hyper-common edge* (a hyper-common edge is defined as an edge that is adjacent to three or more faces). However, they have the same FAG as shown in Figure 4.4d. This suggests that enumerating spanning trees from a FAG may not generate the correct flat layout since there is an ambiguity whenever a feature like that of Figure 4.4a, b or c is present in the folded structure. Fortunately, however, this does not pose a problem as long as the B-rep data of the folded structure is available and utilized in the geometric unfolding/transformation of the faces onto the planar flat layout. This is because the topological information in the B-rep data will correctly identify the common edge that is to be split to unfold the structure. The B-rep data for structures shown in Figure 4.4 is shown in Figure 4.5. Note that only loop tables are shown since they are sufficient to provide needed topology information such as face connectivities and common edge indices. However, this procedure is not entirely workable in the case of hyper-common edges like in Figure 4.4c, and it is a problem that will be examined in more detail in Chapter 5.

Problem II: Different Common Edges Shared by Same Faces

According to the definition of a FAG, each node represents a face of the corresponding 3D structure while each link represents a common edge. However, there may be instances where two faces share more than one common edge. Figure 4.6 illustrates such a problem feature in a structure. For a structure containing such a feature, the links between the problematic faces only represent the connectivity between these faces, rather than represent the specific common edges. Both of the two edges e_1 and e_2 in the 3D structure shown in Figure 4.6a are shared by face 1 and 2. So the link between node 1 and node 2 in the corresponding FAG (Figure 4.6b) actually represents that face 1 and face 2 are connected, instead of representing the edge e_1 or e_2 . Since in the subsequent unfolding stage the FAG link have to provide information about which two faces it con-

Figure 4.4. (a) triangular tubular structure; (b) corner structure; (c) structure with three faces joined at an edge; (d) same FAG for all three structures

loop	face	edges	loop	face	edges	loop	face	edges
l_1	f_1	e_1	l_1	f_1	e_1	l_1	f_1	e_1
		e_2			e_2			e_2
		e_3			e_3			e_3
		e_4			e_4			e_4
l_2	f_2	e_1	l_2	f_2	e_1	l_2	f_2	e_1
		e_5			e_5			e_5
		e_6			e_6			e_6
		e_7			e_7			e_7
l_3	f_3	e_3	l_3	f_3	e_2	l_3	f_3	e_1
		e_8			e_7			e_8
		e_6			e_8			e_9
		e_9			e_9			e_{10}
Structure (a)			Structure (b)			Structure (c)		

Figure 4.5. B-rep data (loop table) for the structures shown in Figure 4.4

Figure 4.6. (a) a structure containing two common edges (e_1 , e_2) shared by the same two faces (1, 2); (b) FAG for the example structure

nects, this is a potential problem because a link (say the link between node 1 and 2 in Figure 4.6b) may indicate any of the common edges (e_1 or e_2) shared by the two faces (1 and 2) as the connection between them. Thus here arises a concern that this ambiguity may lead to different results. However, it is clear that using any of the common edges between two faces as their connection will result in the same flat layout if both of the faces are flat (planar) faces. This is because if the two faces are flat and yet are sharing more than one common edge in the 3D folded structure, then either these two faces are coplanar or their common edges are collinear.

Based on the above discussion, it can be concluded that although the FAG model cannot precisely describe the topology of a 3D structure, it still can be used in the unfolding methodology.

4.2 Flat Layout Generation

To generate the flat pattern that can be folded into a given 3D folded structure, the approach adopted here is by simply generating all possible spanning trees of the corresponding graph of the folded structure. In other words, the graph representation of the flat layout is a spanning tree of the graph of the folded structure. This is consistent with the study made by Lin and Yang (1994) where a “seam line equation” (Equation 3.1) is developed. As a flat layout can be created by unfolding the folded structure, and unfolding that structure means separating a number of previously connected adjacent faces, the

seam line equation helps to answer the question of how many such connections (common edges) need to be broken. The common edges that connect two adjacent faces that are to be separated (in the unfolded layout) are the seam lines while the common edges that remain (in the unfolded layout) are the fold lines. If f is the number of faces, e_c is the total number of common edges in the folded structure and e_s is the number of seam lines, then the seam line equation is:

$$e_s = e_c - f + 1 \quad (4.1)$$

(It may be worthwhile to note that this equation does not apply for non-manifold objects with common edges that join 3 or more faces). If e_f is the number of fold-lines, then e_f should be the difference between e_c and e_s , hence

$$e_f = e_c - e_s = f - 1 \quad (4.2)$$

Therefore the number of common edges in the unfolded flat layout is $f - 1$, and $f - 1$ is also the number of connections in a spanning tree (the minimum number of connections needed to keep all nodes of the graph connected). Since there are no circuits in a tree, generating all possible spanning trees of the graph will produce all possible layouts which are open-chain but not possible layouts that have closed-loop kinematic chains. This means that the flat layouts created can be folded one fold line at a time (whereas layouts with closed-loops may have some portions that have to be folded by folding several fold lines together simultaneously).

4.2.1 Creation of Graph Representation From B-rep Models

Given a B-rep model, we need to create a graph representation for it first. There are two ways to achieve this objective. The first way is based on face connectivity, so it is called *Face Connectivity Algorithm*; the other is based on edge connectivity, thus it is called *Edge Connectivity Algorithm*.

Face Connectivity Algorithm

The graph representation for a given 3D folded structure that has n faces ($F = \{f_i | i = 1, 2, \dots, n\}$) and e common edges ($C = \{c_i | i = 1, 2, \dots, e\}$) can be denoted as $G(F, C)$.

The face connectivity algorithm is described as follows:

```

for all faces  $f_i \in F$  do
  for all faces  $f_j \in F$  (for  $j > i$ ) do
    if exists a common edge  $c_i \in C$  connecting  $f_i$  and  $f_j$  then
      Add  $c_i$  to  $G$ 
    end if
  end for
end for

```

Algorithm 4.1. Face connectivity algorithm

For a structure with F faces, this algorithm has a time complexity of $O(F^2)$. When the number of faces is large, then this algorithm will take too much time.

Edge Connectivity Algorithm

In the B-rep data structure described in Section 4.1.1, there is an item in edge table named *loops belong to* that points to the loops to which each edge belongs. If there is only one loop in this item, then this edge is a free edge; otherwise, it is a common edge. And we can get the indices of the faces that this edge belongs to, which will help us create the graph representation directly.

Algorithm 4.2 illustrates the edge connectivity algorithm.

```

Create  $G(F, \emptyset)$  graph with  $n$  nodes associated with  $n$  faces, and no links.
for all edges  $c_i \in C$  do
  Get the indices of the faces that share  $c_i$  and connect them in  $G$ .
end for

```

Algorithm 4.2. Edge connectivity algorithm

The time complexity for edge connectivity algorithm is $O(E)$. Compared with the time complexity of the former algorithm, the edge connectivity algorithm is more efficient because in most cases, $F < E \ll F^2$. For example, simple polyhedra normally satisfy the Euler's formula, which states that $V - E + F = 2$, where V , E and F denote the number of vertices, edges and faces of the polyhedron, respectively. The edge connectivity algorithm will have the same time complexity as that of the face connectivity algorithm only when $E = aF^2$, where a is a constant coefficient. However, for most structures that are going to be handled in this work, $E \ll F^2$. Other structures have a similar conclusion.

4.2.2 Spanning Tree Enumeration on FAGs

The algorithm described in Shioura et al. (1997) (See Section 2.3.5) was implemented to enumerate all spanning trees of a FAG. The algorithm proposed by Matsui (1997) or Kapoor & Ramesh (1992, 1995) can also be used to enumerate weighted spanning trees sequentially, if the FAG has been assigned weights according to some optimality criteria. This provides a capability to obtain optimized flat layouts in order. However, since the objective of this work is to obtain all non-overlapping flat layouts for a given structure, and the enumeration of unweighted spanning trees is more efficient, therefore Shioura et al. (1997)'s method is used currently.

4.2.3 Geometric Construction of Flat Layouts

In order to generate the flat layout, all faces of the 3D object must be rotated and/or translated to a unique plane. The most straightforward way is to rotate and/or translate each face to the plane. In the example structure shown in Figure 4.7a, after a spanning tree Figure 4.7b is obtained, the plane that contains face 1 can be taken as the unique plane, and all other faces are to be rotated and/or translated to it. Let's label face 1 as the root face. When face 2 is being rotated, face 3 and face 4 must be also under the same rotation, so as to keep the topological properties correct. This means the faces that are directly connected to the root face will be operated once while others have to be

manipulated d times, where d denotes how far a face is from the root.

Figure 4.7. Flat layout generation

Obviously, this method is simple but in most cases is not very efficient. It would have required a huge number of temporary/intermediate results to be generated, so a more efficient alternative algorithm is needed. A way called “Add Face Algorithm” is developed for this purpose. After the root face (1) is selected, the nearest faces (2, 7, 8) can be added to it, according to the topological properties, followed by the second nearest (3), and so on, until all faces are placed. This way, the shortcomings of the previous method can be avoided because each face will be operated upon only once.

Mathematical Formulation of Add Face Algorithm

Figure 4.8 is used to explain the algorithm. Face $abef$ needs to be rotated about axis ab to the plane that is defined by the root face $adcb$, i.e. to the position of $abe'f'$. Face $abef$ and $adcb$ are connected at edge ab . The absolute coordinates of all the vertices of face $adcb$ are known.

Figure 4.8. An example of add face algorithm

A relative reference frame $(e_1^{(0)}, e_2^{(0)}, e_3^{(0)})$ of $abef$ is needed (Figure 4.9), where $e_1^{(0)}$ is the normalized vector of the edge whose destination is known (in this case, it is the normalized vector of ab), $e_3^{(0)}$ is the normal vector of face $abef$, and $e_2^{(0)}$ is the vector product of $e_1^{(0)}$ and $e_3^{(0)}$:

$$\begin{aligned} e_1^{(0)} &= \frac{\vec{ab}}{|\vec{ab}|} \\ e_2^{(0)} &= e_3^{(0)} \times e_1^{(0)} \\ e_3^{(0)} &= \frac{\vec{ab} \times \vec{be}}{|\vec{ab} \times \vec{be}|} \end{aligned} \tag{4.3}$$

Figure 4.9. Relative reference of face

The next step is to get all the relative coordinates of the vertices of the face being operated upon ($abef$), based on the reference frame that was just built (Figure 4.10). Using vertex f to illustrate the method, let $(f_{e_1^{(0)}}, f_{e_2^{(0)}}, f_{e_3^{(0)}})$ denote the relative coordinates of f in reference frame $(e_1^{(0)}, e_2^{(0)}, e_3^{(0)})$. Since $e_3^{(0)}$ is normal to face $abef$, the third

coordinate of f is zero. Therefore,

$$\begin{aligned} f_{e_1^{(0)}} &= |\vec{af}| \cos \theta = \vec{af} \bullet e_1^{(0)} \\ f_{e_2^{(0)}} &= |\vec{af}| \cos(\frac{\pi}{2} - \theta) = \vec{af} \bullet e_2^{(0)} \\ f_{e_3^{(0)}} &= 0 \end{aligned} \quad (4.4)$$

Figure 4.10. Relative coordinates of vertices of face $abef$

Obviously, the relative coordinates of all the vertices of the face will remain unchanged after transformation. So if the (new) changed relative reference frame can be obtained after rotation of $(e_1^{(0)}, e_2^{(0)}, e_3^{(0)})$, we can use coordinate transformation to get the new absolute geometric position of the face. Let the changed relative reference frame be labeled as $(e_1^{(1)}, e_2^{(1)}, e_3^{(1)})$, where $e_1^{(1)}$ is the normalized vector of the changed edge of ab (i.e. $a'b'$), $e_3^{(1)}$ is the normal vector of the destination plane $adcb$, and $e_2^{(1)}$ is the vector product of $e_1^{(1)}$ and $e_3^{(1)}$:

$$\begin{aligned} e_1^{(1)} &= \frac{\vec{a'b'}}{|\vec{a'b'}|} \\ e_2^{(1)} &= e_3^{(1)} \times e_1^{(1)} \\ e_3^{(1)} &= \frac{\vec{ad} \times \vec{dc}}{|\vec{ad} \times \vec{dc}|} \end{aligned} \quad (4.5)$$

Let the absolute reference frame be $(e_1^{(2)}, e_2^{(2)}, e_3^{(2)})$ as shown in Figure 4.11, with

$$e_1^{(2)} = \{1, 0, 0\}, \quad e_2^{(2)} = \{0, 1, 0\}, \quad e_3^{(2)} = \{0, 0, 1\} \quad (4.6)$$

The coordinate transformation matrix to transform coordinates from reference frame

Figure 4.11. Final results after coordinate transform

1 to reference frame 2 is given by:

$$M_{21} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & -a_1^{(2)} \\ a_{21} & a_{22} & a_{23} & -a_2^{(2)} \\ a_{31} & a_{32} & a_{33} & -a_3^{(2)} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{where } a_{ij} = e_i^{(2)} \bullet e_j^{(1)} \quad (4.7)$$

and where $(a_1^{(2)}, a_2^{(2)}, a_3^{(2)})$ are the coordinates of the point a in the absolute reference frame. The final absolute coordinates of vertex f can then be obtained by multiplying the matrix by their original values:

$$\begin{Bmatrix} f_{e_1^{(2)}} \\ f_{e_2^{(2)}} \\ f_{e_3^{(2)}} \\ 1 \end{Bmatrix} = M_{21} \begin{Bmatrix} f_{e_1^{(0)}} \\ f_{e_2^{(0)}} \\ f_{e_3^{(0)}} \\ 1 \end{Bmatrix} \quad (4.8)$$

Based on the above discussion, the add face algorithm can be given in Algorithm 4.3.

Select and place a root face f .

Get normal vector n of f .

for all other faces f_i of the structure **do**

Build up reference frames $(e_1^{(0)}, e_2^{(0)}, e_3^{(0)})$, $(e_1^{(1)}, e_2^{(1)}, e_3^{(1)})$, $(e_1^{(2)}, e_2^{(2)}, e_3^{(2)})$ related to f_i ,

Create Coordinate Transform Matrix M_{21} .

Transform all vertices of f_i .

end for

Algorithm 4.3. The add face algorithm

To compare the time complexity of the add face algorithm with that of the straightforward way mentioned in the beginning of this section, let V_f denote the maximal number of vertices of any face of a given 3D structure and F denote the number of faces. The number of vertices on each face is a random variable which can be any number within a range from 3 to V_f . The corresponding probability for each value of this random variable is $1/(V_f - 2)$. Thus, the average number of vertices on each face is the expectation of this random variable:

$$\begin{aligned} V_{ave} &= \frac{1}{V_f - 2} + \frac{2}{V_f - 2} + \frac{3}{V_f - 2} + \dots + \frac{V_f}{V_f - 2} - \frac{1}{V_f - 2} - \frac{2}{V_f - 2} \\ &= \sum_{i=1}^{V_f} \frac{i}{V_f - 2} - \frac{3}{V_f - 2} \\ &= \frac{V_f^2 + V_f - 6}{2(V_f - 2)} \end{aligned} \quad (4.9)$$

So on average, for each operation of adding a face will cost $O(V_f)$ time. Thus the add face algorithm has a total time complexity of $O(FV_f)$. In the straightforward algorithm, rotating and/or translating a face will cost $O(V_f)$ time as well. However, at i^{th} time when flattening a face will need some other faces to rotate and/or translate together with it. Rotation and/or translation times will be determined by the height of the spanning tree. Assuming that the height of a spanning tree is H and that there are n_i nodes below the i^{th} level of the spanning tree. Thus the total rotation and/or translation time T is determined by the following:

$$T = \sum_{i=1}^H n_i \quad (4.10)$$

and subsequently the total time cost of the algorithm is $O(\sum_{i=1}^H n_i V_f)$. This shows how much more efficient is the add face algorithm compared to the straightforward one.

4.2.4 Orientation of Faces

The faces of both manifold and non-manifold structures can be looked upon as two-sided. The two sides have opposite orientations and opposite normal vectors.

For example, in Figure 4.12, assume that face π is going to be flattened to the plane determined by the dashed line segments. If n_2 is used as the normal vector, π_1 is ob-

Figure 4.12. Orientation of faces

tained; however, if n_2' is used as the normal vector, then π_1' is obtained. Thus, when flattening faces, different positions may be obtained when different normal vectors are selected.

However, if the flattening face has a parent (the face that the flattening face is connected to) and a non-overlapping result is desired, only one correct normal vector can be utilized. For example, assume that face π is going to be flattened to the plane that is determined by face τ (shown in Figure 4.13). If n is selected as the normal vector of π , then a correct result (π_1 and τ) is obtained; if n' is selected, π will overlap with τ after it is flattened (π_1'). Thus if the orientations of their parents are determined, then their orientations are also determined. The correct orientation can be identified easily. If a face and its parent are using opposite orientations at the common edge (in this case the common edge is the edge ab), then the orientations indicated by this edge will lead to the correct normal vector and hence the correct non-overlapping result.

4.2.5 Overlapping Detection

Not all the layout designs generated from the spanning trees are necessarily valid or usable because some may contain faces which overlap. An overlapping detection method is presented here to check and hence eliminate any such designs in the final results. The boundaries of the faces of the 3D structures in this work are all simple polygons. A polygon P is said to be simple if the only points of the plane belonging to two edges of P are the vertices of P . This definition indicates that two edges of a simple polygon that

Figure 4.13. Selection of the normal vectors for faces

are not adjacent must not intersect each other.

There are four possibilities in the relative positions of any two given faces, as shown in Figure 4.14. Two faces are overlapping if at least one interior point of a face is an interior or a boundary point of the other. According to this definition, two faces are not overlapped if they are only adjacent (connected at the boundaries) (Figure 4.14b).

Figure 4.14. Possible relative positions of faces

Intersection Detection Algorithm

Intersection detection between two simple polygons is a fundamental research field of computational geometry. A well-known method known as the “sweep-line algorithm” (discussed in Appendix B) is used as the foundation of our overlapping detection algorithm.

Given two faces f_1 and f_2 , the first step is to obtain the approximate relationship between them. Let $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ denote the leftmost vertices of f_1 and f_2 ,

respectively and let

$$P = \begin{cases} P_1 & \text{if } x_1 \geq x_2 \\ P_2 & \text{if } x_1 < x_2 \end{cases}$$

and f be the other face that does not contain P . A point location query (discussed in Appendix B) of P against f is then executed. If P is contained in f , then f_1 and f_2 overlap; otherwise, proceed to the next step of detecting intersections among the edges of f_1 and f_2 by using the sweep-line technique.

Once an intersection is detected, it still cannot be asserted that f_1 and f_2 overlap. This is because of the three main styles of intersections, which are shown in Figure 4.15.

For convenience, let us say that line segments ab and cd belong to face f_1 and f_2 , respectively. If the intersection is with style (a), an overlap happens. Obviously f_1 lies on either end of ab . However because cd crosses both sides of ab , there must be at least one point on cd located inside of f_1 , which means f_1 and f_2 overlap.

Figure 4.15. Possible relative positions of intersecting lines

Style (b) is more complicated because f_1 and f_2 may be either overlapped or just simply connected under this situation. Moreover there are two detailed sub-styles for style (b), i.e. ab and cd are partly or wholly coincident. In fact it is not necessary to consider this intersection style because if (b) happens then style (c) happens as well, thus consideration of (c) is enough for both types of situations. However, efficiency of overlapping detection can be improved by utilizing the orientation of edges in style (b).

An example is given in Figure 4.16 to illustrate face relationships when they share coincident edges. f_1 is a face that contains an inner ring. The orientation of faces is assumed as: when walking along the boundary of a face following its orientation, the face should always be on the left side. According to this definition, the outer boundary

and inner boundary have opposite orientations as shown in Figure 4.16. f_2 and f_3 share edges ab and cd with the outer boundary of f_1 , respectively, while f_4 and f_5 share edges ef and gh with the inner boundary of f_1 , respectively. gh is partly coincident with ef . If two faces share a common edge e , e can be the outer boundary of these two faces (as in the case of ab and cd) or the outer boundary of one face and the inner boundary of the other (as in the case of ef and gh). The former common edges are named *outer boundary common edges* and the latter are *inner boundary common edges*. If two faces are connected at a common edge e and have the same orientation at e (e.g. f_1 and f_3 (outer boundary common edge case) or f_1 and f_5 (inner boundary common edge case)), then they overlap. Determination of this kind of overlapping will cost only $O(1)$ time. Other situations such as faces having opposite orientations at a common edge will be addressed in the management of style (c).

Figure 4.16. Face relationships when sharing a common edge

Intersection style (c) has two special cases as well. The first case is where one endpoint of a line segment lies on the other line segment other than its two endpoints (Figure 4.17a); the second case is where one endpoint of a line segment coincides with the other line segment's endpoint (Figure 4.17b).

In Figure 4.17a, endpoint c of line segment cd lies on ab , but does not coincide exactly with a or b . The orientation of ab is indicated by the thick arrow and the gray part indicates the face f that contains edge ab . Obviously, some parts of cd is located in

f and hence the face that contains cd overlaps f . Let $\vec{n} = \vec{ca} \times \vec{cd}$, and if the orientation of vector n coincides with that of f , then it can be asserted an overlapping has occurred. This is because line segment cd and f are both on the same side of ab . This procedure will take $O(1)$ time.

Figure 4.17. Two special cases for intersection style (c)

In Figure 4.17b, assume that line segment ab intersects ad at point a , which is the endpoint for both ab and ad . f_1 and f_2 are the faces that contain ab and ad , respectively. Let ac be the edge that is connected to ab at a in f_1 , and ae be the edge that is connected to ad at a in f_2 . θ_1 is the angle from ab to ac and θ_2 is the angle from ab to ad . If $\theta_1 < \theta_2$ then ac lies outside of f_2 ; otherwise f_1 and f_2 overlap. Similarly, θ_3 is the angle from ad to ae and θ_4 is the angle from ad to ab (i.e. $\theta_4 = 2\pi - \theta_2$). If $\theta_3 < \theta_4$ then ab lies outside of f_2 ; otherwise f_1 and f_2 overlap.

4.2.6 Methodology Outline

An overall methodology based on spanning tree enumeration was developed. Algorithm 4.4 outlines this methodology.

```

Construct the corresponding FAG  $G$  for the 3D structure. {See Algorithm 4.2}
repeat
    Enumerate next spanning tree  $T_i$  from  $G$ .
    Construct flat layout  $L_i$  by using  $T_i$ . {See Algorithm 4.3}
    if  $L_i$  is not overlapped then
        Output  $L_i$ .
    end if
until no more spanning trees in  $G$ .

```

Algorithm 4.4. The unfolding methodology based on spanning tree enumeration

4.2.7 Time Complexity Analysis of Methodology

From Algorithm 4.4, it can be seen that the algorithm consists of four parts, i.e. FAG construction, spanning tree enumeration, flat layout construction and overlapping detection. The time complexity for each part has been discussed in the previous sections and these results are used here to determine the time complexity of the overall methodology. Assume that the time costs in constructing FAG, enumerating spanning trees, constructing a flat layout and detecting overlapping within a flat layout are T_f , T_e , T_l and T_o , respectively. The overall time expense is:

$$T = T_f + T_e + S(T_l + T_o)$$

Suppose the 3D structure has F faces and E edges. The maximum number of vertices within a single face in the structure is V_f . The corresponding FAG has N nodes, L links and S spanning trees. From previous sections, $T_f = O(E)$, $T_e = O(N + L + S)$, $T_l = O(FV_f)$ and $T_o = O(E \log(E))$ (see Appendix B). It should be noted that $N = F$. Therefore, the overall time complexity is:

$$T = O(E + F + L + S(FV_f + E \log(E))) \quad (4.11)$$

4.2.8 Methodology Illustration

An example is shown in Table 4.1 to illustrate the result of the methodology. The only complication of this example is that there is a depression on the top face of the structure. In order to make it unfoldable, the top face was manually split into two separated faces (i.e. face 1 and 2). Among the 13,178 found flat layouts, four were selected and listed in the table. Note that the 3D structure FAG and spanning trees for the respective layouts have also been given.

Table 4.1. Methodology Illustration

<div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>

Total Spanning Trees of the Original FAG	172,032
Total Non-overlapping Flat Layouts	13,178
CPU Time (s)	605.437
<div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> </div> <div style="text-align: center;"> </div> </div>	
<i>continued on next page</i>	

continued from previous page

(a)

(b)

(c)

(d)

4.3 Other Aspects of Unfolding

In this section, a few other issues related to unfolding procedure will be analyzed and solutions will be given.

4.3.1 Depression, Protrusion and Through-hole Features

Most of the structures that contain depressions, protrusions or through-holes do not have any edge unfolding. This is mainly due to the fact that the faces that make up a depression, protrusion or through-hole cannot be unfolded within the space enclosed by the ring of the connecting face (i.e. the face which is connected to this feature). In order to get a single piece flat layout, the structure has to be modified prior to unfolding. The modification method is to split the connecting face into two or more faces so that the interior ring is also split to become parts of the exterior loops of the newly created faces.

Figure 4.18. *Depression treatment*

For example, Figure 4.18a shows a structure containing a depression. Without modification, the structure has no non-overlapping edge unfoldings. By adding two extra edges (the thin solid lines on the top face), the top face is split into two and as a result, the structure can be unfolded into a non-overlapping flat layout (shown in Figure 4.18b). The treatment for protrusions or through-holes is similar to this example.

The structure modification is assumed to be done by the designer manually. This is because, in most cases, the task of determining how to split connecting faces is best left to the designer, rather than let it be handled by computers. The modification of a given 3D structure should be minimal and just enough to ensure edge unfolding is possible.

4.3.2 Face Merging Problem

The tube-like structure shown in Figure 4.19a can possibly be unfolded in the manner shown in Figure 4.19b. The unfolding strategy is simple but it is difficult for a computer program to automatically recognize this possibility. For example, can the top face be considered as one face or as two faces that are connected at a common edge? In this case, the treatment seems to handle it as one face. A similar problem is shown in Figure 4.19c, where the leftmost face is treated as two individual faces while some other faces seem to have been merged in Figure 4.19d.

In a given 3D structure, if there exists two adjacent faces that share the same normal

Figure 4.19. *The face merging problem*

vectors, this problem arises. Since these faces are coplanar in the 3D structure, the common edges which they share can be ignored if desired. So these faces can be viewed as one single face. To treat them as one single face has an advantage in that the number of nodes and links will be reduced in the corresponding FAG. This means the number of spanning trees will be greatly reduced. However, on the other hand, as there may be some other faces connected at the common edges that the two faces are sharing, some valid flat layouts may be missed if those two faces are treated as one.

Therefore, in order not to miss out on any valid flat layout, all the faces involved in this problem will be considered as separate individual faces. However, if such faces are still connected in any given flat layout, the common edge between them can then be removed and the two faces joined as one without a folding line in between. It will make it look like the two faces were treated as one (just like the top and leftmost faces of Figure 4.19b).

4.3.3 Optimality Criteria

The objective of flat layout enumeration is usually to find the optimal result. However, in most cases, there may not be a unique optimal flat layout according to different opti-

maximal points of view. Therefore, all non-overlapping flat layouts are enumerated by the methodology and saved for further assessments. In this section, a few optimality criteria will be discussed using the L-shape structure example shown in Figure 4.20a. Figure 4.20b is the corresponding FAG for this structure.

Figure 4.20. Optimality criteria example: an L-shape and its corresponding FAG

Topological Compactness

Topological compactness can be measured by the sum of eccentricities (Wang, 1997). The eccentricity of a node in a tree is defined as the distance (the shortest path between two nodes) between this node and the farthest node from it (Deo, 1974). The topological compactness is the inverse of the sum of eccentricities of all nodes in a spanning tree. However, the most topologically compact flat layout may not necessarily be useful in practice since it may not be good according to geometric optimization criteria (e.g. geometric compactness). Therefore, topological compactness criterion is used only when there is a difficulty in differentiating some flat layouts that are equally good according to some geometric criteria.

Geometric Compactness

A measure of geometric compactness is the ratio of the area to the square of the perimeter (Wang, 1997). This definition is similar to that presented by Ballard & Brown (1982). Since every non-overlapping flat layout for a given 3D structure must have the same area, geometric compactness is equivalently and simply measured by the perimeter of the

layout. The most geometrically compact layout is the one with the minimum perimeter. This also implies that longer connecting edges (folding lines in the layout) are preferable to shorter connecting edges. The most geometrically compact layout can make handling of the layouts easier and the folded 3D structure more robust, since the folding lines are relatively longer than in other layouts. Figure 4.21a shows the most geometrically compact flat layout for the L-shape example structure of Figure 4.20.

Minimum Enclosing Area

Another desired objective is to obtain the flat layout with the minimum enclosing area, which is the area of the smallest rectangle that contains the layout. Quite often, the flat layout with minimum enclosing area is the most cost efficient in terms of cutting the layouts from the stock material (sheet) and in terms of storage/transportation of the layouts. The layout with the minimum enclosing area is not always the most geometrically compact, although sometimes it is. This is because the layout with the minimum enclosing area does not always have the minimum perimeter. Figure 4.21b shows the layout with the minimum enclosing area. It is obviously different from the most geometrically compact layout shown in Figure 4.21a. So for minimizing costs, the minimum enclosing area may be the most important consideration.

X, Y and Overall Extent

There may sometimes be demands to find a layout within a specified extent in the x or y direction, or perhaps a layout with the minimum x or y extent. This is because the dimensions of the raw material for making the 2D flat layouts may be predefined. Therefore, flat layouts with specified x or y extent may be the best choice based on considerations of both producibility and material costs/efficiency. For the above example, the two flat layouts shown in Figure 4.21d and e are the results with minimum x and y extent, respectively.

The overall extent of a flat layout is the larger of the two extents in the x and y directions. Another applicable demand is to obtain the flat layout that has the minimum

overall extent since this layout is compact in both the x and y direction (this is because the other extent is even smaller, according to the definition of the overall extent). The minimum overall extent for the example is shown in Figure 4.21c.

Interestingly, although geometric compactness, minimum enclosing area and minimum overall extent seem to have similar objectives (i.e. to find compact flat layouts), different results may be obtained according to different criteria (as can be seen from Figure 4.21).

Figure 4.21. Different optimal results based on different optimality criteria.

CHAPTER 5

Unfolding with Topological Recognition

The spanning tree enumeration methodology discussed in Chapter 4 is theoretically simple and straightforward. However, complications in some non-manifold structures, like those that contain hyper-common edges, pose problems to the methodology that can lead to incorrect results. This chapter describes the problems caused by non-manifold structures and develops a modified spanning tree enumeration methodology with an additional topology recognition module. The modified methodology can correctly unfold structures with complications (which cannot be handled by the previous method), and can improve the overall enumeration efficiency.

5.1 Hyper-Common Edge

5.1.1 Definitions

Hyper-Common Edges A hyper-common edge is a common edge that is adjacent to three or more faces. If the i^{th} edge is a hyper-common edge, it is denoted as e_i^h .

Hyper-Common Edge Set and its Sequence The hyper-common edge set of a specific hyper-common edge is the set of all faces adjacent to this hyper-common edge. The sequence of a hyper-common edge set is a sorted hyper-common edge set. The first element in the sequence can be any of those faces arbitrarily selected. However, the subsequent elements are all the remaining faces sorted in either clockwise or counter-clockwise order. For a hyper-common edge e_i^h , its hyper-common edge set is denoted as $H(e_i^h)$ and its sequence is denoted as $S(e_i^h)$.

5.1.2 Problems Arising from Hyper-Common Edges

Problem I: Incomplete Splitting of Hyper-Common Edge

Figure 5.1. (a) structure with hyper-common edge; (b) topologically valid spanning tree; (c) topologically invalid spanning tree.

The presence of hyper-common edges in a folded structure poses a problem to the basic procedure of obtaining a flat layout via generating a spanning tree from the FAG of the structure. This is illustrated in Figure 5.1a which shows a folded structure featuring one hyper-common edge e_0^h (with hyper-common edge set $H(e_0^h) = \{1, 2, 3\}$), with Figure 5.1b showing one possible spanning tree that gives rise to a correct flat layout (the dashed lines show the links that are removed from the original FAG of the folded structure to reduce it to a spanning tree). However, Figure 5.1c also shows a spanning

tree but one that is topologically invalid because it represents an erroneous situation where the hyper-common edge has not been split completely since it is impossible for both faces 1 and 3 to be adjacent to face 2 and yet are not adjacent to each other. It will therefore not be an acceptable result for a flat layout.

Problem II: Incorrect Splitting of Hyper-Common Edge

Figure 5.2. (a) structure with hyper-common edge; (b) corresponding FAG; (c) a flat layout that cannot be folded into structure; (d) topologically invalid spanning tree.

If a hyper-common edge set contains four or more faces, the sequence of the set is a factor to be considered because it restricts the possible ways in which the hyper-common edge can be split up. Consider the structure shown in Figure 5.2a featuring a hyper-common edge e_0^h with the sequence of faces $S(e_0^h) = \{2, 5, 3, 6\}$. The structure has a total of six faces with a FAG as shown in Figure 5.2b. Figure 5.2c shows one flat layout with its corresponding spanning tree shown in 5.2d. However, this layout cannot be folded into the 3D structure because there will be a clash/intersection among the edges where the hyper-common edge used to be since the adjacent faces 5 and 6 have

to pass through adjacent faces 2 and 3. In other words, not every spanning tree leads to a layout that can be folded in the required manner because of the possibility of incorrect splitting arrangement among the hyper-common edge set. Such incorrect trees are also considered topologically invalid.

5.1.3 Solution to Hyper-Common Edges

The following section will describe the solution methodology developed for solving the problems caused by hyper-common edges. It includes the computational algorithms developed to automatically determine the sequence of a hyper-common edge set, and the procedures to detect topologically valid/invalid spanning trees at locations where the hyper-common edges used to be. The sequence of the set has to be determined because this information is necessary for checking the validity of the spanning trees.

Determining the Sequence

Consider a hyper-common edge denoted by the vector e_0^h and its corresponding edge set $H(e_0^h) = \{1, 2, 3, 4\}$ consisting of four faces and shown in Figure 5.3. The sequence $S(e_0^h)$ of this set is to be determined automatically. The sense of each of the unit normal vectors n_1 to n_4 of the faces of this set can be consistently defined (as shown in Figure 5.3) since all of the faces share the same edge e_0^h and the direction of e_0^h is fixed. It should also be noted that all normal vectors of the faces sharing the same hyper-common edge can be translated to a position where they become coplanar. This can be proven by moving all the normal vectors onto the hyper-common edge as well as onto the same point along that edge. Since all the normal vectors must also be perpendicular to that edge, it follows that they now lie in the same plane. Since all the normal vectors are coplanar, those normal vectors from Figure 5.3 can be drawn on the same plane with their corresponding orientations with respect to one another as shown in Figure 5.4.

Based on the angles between the normal vectors, the sequence of the faces can be deduced. Any one of the faces can be chosen as the start of the sequence and in this illustration, face 1 is chosen. The angles (as shown in Figure 5.4 between face 1 and any

other face i can be computed as follows:

$$\theta_{1i} = \begin{cases} \cos^{-1}(n_1 \bullet n_i) & \text{if } (n_1 \times n_i) \bullet e_0^h \geq 0 \\ 2\pi - \cos^{-1}(n_1 \bullet n_i) & \text{if } (n_1 \times n_i) \bullet e_0^h < 0 \end{cases} \quad (5.1)$$

According to Equation 5.1, each angle θ_{1i} will correctly range from 0 to 2π . The sequence can then be determined by sorting according to the size of the angles and, in this illustration, the sequence is of course $S(e_0^h) = \{1, 2, 3, 4\}$.

Figure 5.3. Faces of a hyper-common edge set.

Figure 5.4. Orientation of normal vectors of the hyper-common edge set.

Detecting Topological Validity

Consider a hyper-common edge set consisting of six faces with a sequence $S(e^h) = \{1, 2, 3, 4, 5, 6\}$ as shown in Figure 5.5a together with its FAG. Note that this set may be just one part of a larger structure. Figure 5.5b shows one possible way in which the hyper-common edge has been split and its corresponding spanning tree (actually only a portion of the overall layout's spanning tree because the nodes and links related to the

rest of the structure are not shown). Notice that Problem I (Section 5.1.2) is present in this result because there is still a hyper-common edge set $\{2, 3, 6\}$ remaining, while Problem II (Section 5.1.2) is also present because adjacent faces 1 and 4 have to intersect adjacent faces 2, 3 and 6 if the layout is to be folded into the structure. Such a spanning tree will therefore be considered invalid.

A possible valid way of splitting the hyper-common edge is shown in Figure 5.5c together with its corresponding spanning tree (note that face 1 and face 3 are not connected to any other faces from among the hyper-common edge set but may instead be connected to some other faces from the rest of the structure not shown). To detect the validity of any given spanning tree, the nodes and links pertaining to any particular hyper-common edge set $H(e^h)$ can first be placed along a circle in an order according to its sequence $S(e^h)$, with all links drawn as straight lines. To illustrate, the valid spanning tree from Figure 5.5c (which is already in its sequence) is placed within the dashed circle as shown in Figure 5.6. Based on such an arrangement, it is now claimed that any spanning tree is valid only if the following two conditions are satisfied:

1. No node along the circle is incident on more than one link.
2. No link intersect any other link within the circle.

The first condition eliminates Problem I because any node that is incident on two or more links represents an incomplete splitting of the hyper-common edge. Note that this condition also implies that, in a valid spanning tree, the maximum allowable number of links among the hyper-common edge set is c_{max} , given by

$$c_{max} = \left\lfloor \frac{n(e^h)}{2} \right\rfloor \quad (5.2)$$

where $n(e^h)$ is the number of faces in the hyper-common edge set and $\lfloor \cdot \rfloor$ is the floor function. The second condition eliminated Problem II because any intersection of links represents an incorrect splitting (i.e. an intersection of faces when folding). The algorithm to detect this condition can be based on traversing each node in sequence and applying a form of last-in-first-out mechanism to check the links incident to each node.

If a structure has more than one hyper-common edge, then both conditions must be satisfied for each and every such edge before a spanning tree is certified valid.

Figure 5.5. (a) hyper-common edge set and FAG; (b) invalid splitting arrangement and spanning tree; (c) valid splitting arrangement and spanning tree.

Figure 5.6. Topologically valid spanning tree.

5.2 Topological Recognition with Overlapping Paths

In Chapter 4, each spanning tree is assumed topologically valid and used as the topology information for the geometric construction of its corresponding flat layout. The layout is then considered valid only when all faces do not overlap with one another.

Therefore, in order to obtain all valid flat layouts, a 3D structure has to be unfolded and tested for overlapping as many times as the number of spanning trees the corresponding FAG has. However, in most cases, geometric unfolding and overlapping detection are time-consuming processes, especially when the numbers of faces and common edges are large. This section will explore some avenues to avoid these inefficient steps.

In this section, the idea of topology recognition used in our work will be explained. Topology recognition is based on the concept of an overlapping path (which is defined as a path in the corresponding FAG that will lead to an overlapped layout if it exists in a spanning tree). Beginning with the simplest case of overlapping paths, this section will explain step by step why overlapping paths can be used to recognize invalid spanning trees and how they are identified.

5.2.1 Inadmissible Links

Figure 5.7a shows a simple structure with a protrusion at a corner of face 1. Figure 5.7b is a spanning tree of the corresponding FAG for this structure and Figure 5.7c is the unfolded flat layout generated from that spanning tree. It can be seen that there is overlapping between faces 1 and 2 in the layout since the two faces are directly connected (i.e. there is a link between node 1 and 2). Moreover, it can be easily deduced that any spanning tree containing this link will result in an invalid flat layout with overlapped faces 1 and 2. This is because if the link exists, face 2 will always be adjacent to face 1. Therefore, an overlapped situation similar to that shown in Figure 5.7c will always occur. According to the definition of overlapping paths, node 1 and 2, together with the link (1, 2), form an overlapping path with distance of 1. The link (1, 2) (and all of other similar links) are defined as *Inadmissible Links*. An inadmissible link l_{ij}^{in} refers to a link between node i and j in the FAG of a specific 3D structure such that if it is present in any spanning tree, the resulting layout will surely contain overlapping.

The management of inadmissible links can be very straightforward. Since any inadmissible link in a FAG will cause overlapping, it can be simply deleted from the FAG without the risk of missing out on any valid flat layout. Normally, deleting any link from

Figure 5.7. (a) a simple structure; (b) a spanning tree of the FAG for the structure; (c) the corresponding flat layout.

the FAG will greatly reduce the total number of spanning trees. This can be illustrated by using the method for calculating the number of spanning trees introduced in Section 2.3.4. Let D_0 be the degree matrix for the original FAG from which an inadmissible link l_{ij}^{in} is going to be deleted. Similarly, let D_1 be the degree matrix for the FAG after l_{ij}^{in} has been deleted. The main focus is put on those different elements between D_0 and D_1 , i.e. elements affected by both i and j .

$$D_0 = \begin{bmatrix} \ddots & \vdots & \ddots & \vdots & \ddots \\ \dots & d_i & \dots & -1 & \dots \\ \ddots & \vdots & \ddots & \vdots & \ddots \\ \dots & -1 & \dots & d_j & \dots \\ \ddots & \vdots & \ddots & \vdots & \ddots \end{bmatrix}, D_1 = \begin{bmatrix} \ddots & \vdots & \ddots & \vdots & \ddots \\ \dots & d_i - 1 & \dots & 0 & \dots \\ \ddots & \vdots & \ddots & \vdots & \ddots \\ \dots & 0 & \dots & d_j - 1 & \dots \\ \ddots & \vdots & \ddots & \vdots & \ddots \end{bmatrix}$$

By deleting the i_{th} row and column from both D_0 and D_1 , the number of spanning trees for the corresponding two FAGs can be obtained by calculating the determinants of the remaining submatrices, as indicated by Equation 5.3, where N_0 and N_1 denote the number of spanning trees for the original FAG and modified FAG, respectively.

$$N_0 = \begin{vmatrix} \ddots & \vdots & \ddots \\ \dots & d_j & \dots \\ \ddots & \vdots & \ddots \end{vmatrix}, N_1 = \begin{vmatrix} \ddots & \vdots & \ddots \\ \dots & d_j - 1 & \dots \\ \ddots & \vdots & \ddots \end{vmatrix} \quad (5.3)$$

Therefore, the reduction in the number of spanning trees is given by

$$N_r = N_0 - N_1$$

It is clear that N_r is the value of the determinant resulting from deleting the affected rows and columns (i.e. i^{th} and j^{th}) from D_0 . Taking the example shown in Figure 5.7, the corresponding degree matrix is:

$$D_0 = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 6 & -1 & 0 & -1 & -1 & -1 & -1 \\ -1 & -1 & 4 & 0 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & -1 & -1 & 0 \\ -1 & -1 & 0 & -1 & 4 & 0 & -1 & 0 \\ 0 & -1 & -1 & -1 & 0 & 5 & -1 & -1 \\ 0 & -1 & 0 & -1 & -1 & -1 & 4 & 0 \\ 0 & -1 & -1 & 0 & 0 & -1 & 0 & 3 \end{bmatrix}$$

And the affected rows and columns are the 1st and 2nd, thus the reduced number of spanning trees is:

$$N_r = \begin{vmatrix} 4 & 0 & 0 & -1 & 0 & -1 \\ 0 & 4 & -1 & -1 & -1 & 0 \\ 0 & -1 & 4 & 0 & -1 & 0 \\ -1 & -1 & 0 & 5 & -1 & -1 \\ 0 & -1 & -1 & -1 & 4 & 0 \\ -1 & 0 & 0 & -1 & 0 & 3 \end{vmatrix} = 1860$$

The number of spanning trees for the original FAG was 4980, therefore there was a reduction by more than 37%.

5.2.2 Exclusive Link Sets

Although the deletion of inadmissible links can greatly reduce the number of spanning trees, inadmissible links do not appear that frequently. In fact, there are many 3D structures that do not contain any inadmissible link at all but still result in many overlapped flat layouts, even for convex structures. It can be discovered from observation that some invalid flat layouts may contain the same overlapping path. For example, Figure 5.8a shows a non-manifold L-shape. Figure 5.8b is its corresponding FAG and Figure 5.8c and e are two overlapped flat layouts for this structure. These two layouts have the same overlapping path $\{1, 4, 2\}$ as seen in their respective spanning trees in Figure 5.8d and f. Likewise, any flat layouts containing this overlapping path will surely be invalid. Therefore, if all overlapping paths within the 3D structure are identified in advance, overlapped

spanning trees with overlapped faces can be detected by detecting the existence of any overlapping path in it. However, enumeration of all overlapping paths can be even more time-consuming than enumerating spanning trees since the number of paths within a graph is far greater than the number of spanning trees. Therefore, the search for overlapping paths is constrained to those that has a maximum distance of 2 (i.e., the path contains only 2 links). It should be noted that an overlapping path with a distance of 1 represents ‘local overlapping’ while an overlapping path with a distance of 2 or more represents a form of ‘global overlapping’.

Figure 5.8. Illustration of overlapping paths.

Figure 5.9a shows a cube like structure. There is a depression on the top face, and the bottom face of the cube was removed. Figure 5.9b is the FAG for this structure. Figure 5.9c and d are an overlapping flat layout and its corresponding spanning tree. It can be seen that the path $\{4, 3, 2, 6, 8, 9, 7\}$ is an overlapping path with distance of 7. Any specific spanning tree will lead to an overlapped flat layout if this path exists in it.

For the example shown in Figure 5.8, all overlapping paths with maximum distance of 2 are listed below:

$$\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 2\}, \{1, 4, 2\}, \{2, 1, 3\}, \{2, 1, 4\}, \{3, 1, 4\}, \{3, 2, 4\}.$$

Figure 5.9. An example of overlapping path with distance of 7.

They also can be listed as following link sets:

$$\begin{aligned} &\{(1, 2), (2, 3)\}, \{(1, 2), (2, 4)\}, \{(1, 3), (3, 2)\}, \{(1, 4), (4, 2)\}, \\ &\{(2, 1), (1, 3)\}, \{(2, 1), (1, 4)\}, \{(3, 1), (1, 4)\}, \{(3, 2), (2, 4)\}. \end{aligned}$$

A link (i, j) is said to be *exclusive* to another link (j, k) if $\{i, j, k\}$ is an overlapping path. An *exclusive link set* $E(i, j)$ for any link (i, j) is defined as a set of all those links from the corresponding FAG that are exclusive to (i, j) . According to this definition, the exclusive link set for each link in Figure 5.8b can be listed as follows:

$$\begin{aligned} E(1, 2) &= \{(1, 3), (1, 4), (2, 3), (2, 4)\} & E(1, 3) &= \{(1, 2), (1, 4), (2, 3)\} \\ E(1, 4) &= \{(1, 2), (1, 3), (2, 4)\} & E(2, 3) &= \{(1, 2), (1, 3), (2, 4)\} \\ E(2, 4) &= \{(1, 2), (1, 4), (2, 3)\} & E(3, 5) &= \emptyset \\ E(3, 6) &= \emptyset & E(4, 5) &= \emptyset \\ E(4, 6) &= \emptyset & E(5, 6) &= \emptyset \end{aligned}$$

It can be seen that each overlapping path is duplicated once in the list. To improve efficiency, it is necessary to eliminate this repetition by comparing every two links in the

undirected graph. Let $l_1 = (i, j)$ where $i < j$ and $l_2 = (m, n)$ where $m < n$. l_1 and l_2 are two links of a given undirected graph $G(N, L)$ such that $l_1 < l_2$ if $i < m$ or for the case $i = m$, $l_1 < l_2$ if $j < n$. The exclusive link set definition can be further confined such that any link in an exclusive link set associated with a specific link l_i should be greater than l_i . Therefore, the exclusive link sets for the above example are adjusted to:

$$E(1, 2) = \{(1, 3), (1, 4), (2, 3), (2, 4)\} \quad E(1, 3) = \{(1, 4), (2, 3)\}$$

$$E(1, 4) = \{(2, 4)\} \quad E(2, 3) = \{(2, 4)\}$$

$$E(2, 4) = \emptyset \quad E(3, 5) = \emptyset$$

$$E(3, 6) = \emptyset \quad E(4, 5) = \emptyset$$

$$E(4, 6) = \emptyset \quad E(5, 6) = \emptyset$$

Hence there is no repetition of overlapping paths.

5.3 Methodology with Topological Recognition

5.3.1 Methodology Outline

Based on the above discussion, there arises a possibility that an overlapped or topologically invalid flat layout can be predicted by just examining its corresponding spanning tree. This will reduce the time wasted in geometrical construction of invalid flat layouts, as well as in overlapping detection. A modification of the methodology derived in Chapter 4 is proposed to implement this possibility. The following algorithm outlines this modified method.

In order to perform a topology recognition on each spanning tree obtained from a spanning tree enumeration process, the topology validity information must be obtained/constructed before the enumeration starts. Therefore, hyper-common edge sequences, inadmissible links and exclusive link sets must be found in the very beginning of an enumeration procedure. A spanning tree will be used to construct a flat layout only when it does not contain any invalid hyper-common edge topologies, inadmissible links or overlapping paths.


```

Construct the corresponding FAG  $G = (N, L)$  for the structure. {See Algorithm 4.2}
for all edges  $e_i$  of the structure do {Construct all hyper-common edge sequences.}
  if  $e_i$  is a hyper-common edge then
    Construct  $S(e_i^h)$  for  $e_i$ .
  end if
end for

for all faces  $f_i$  of the structures do {Find inadmissible links and exclusive link sets.}
  Unfold each face  $f_j$  that is adjacent to  $f_i$  to the plane defined by  $f_i$ .
  if  $f_j$  overlapped with  $f_i$  then
    Delete the link  $(i, j)$  from  $G$  {(i, j) is an inadmissible link.}
  else if  $f_j$  overlapped with another just unfolded face  $f_k$  which is adjacent to  $f_i$  then
    Update exclusive link sets by adding a new overlapping path  $\{j, i, k\}$ .
  end if
end for

repeat
  Enumerate next spanning tree  $T^c$  from  $G$ .
  if  $T^c$  satisfies both hyper-common edge validity check and exclusive link set validity check then
    Construct flat layout  $L^c$  by using  $T^c$ . {See Algorithm 4.3}
    if  $L^c$  is not overlapped then
      Output  $L^c$ .
    end if
  end if
until no more spanning trees in  $G$ .

```

Algorithm 5.1. Modified Unfolding Methodology

5.3.2 Time Complexity Analysis

To analyze the time complexity of the methodology, some input information related to a specific 3D structure must be known. Let V , E , E_h , F be the number of vertices, common edges, hyper-common edges and faces, respectively. Let N , L and S denote the number of nodes, links and spanning trees of the graph modified from original FAG by deleting inadmissible links. Furthermore, let F_h be the maximum number of faces that share a hyper-common edge and V_f be the maximum number of vertices within a single face.

Basically, the methodology contains two parts: the first part is to analyze the model so as to collect all the necessary topology information (i.e., hyper-common edge sequences, inadmissible links and exclusive link sets). The second part is to enumerate all the spanning trees of the graph that was modified from the original FAG by deleting all the inadmissible links. Each spanning tree obtained needs to be examined on both hyper-common edge validity and exclusive link set validity. Valid spanning trees will be further processed for constructing geometric flat layout, which will in turn be checked for overlapping. Therefore, the total time cost can be calculated based on the following equation:

$$T = T_h + T_e + T_l \quad (5.4)$$

where T is the total time cost. T_h , T_e , T_l are the time cost of finding hyper-common edge sequences, finding inadmissible links and exclusive link sets, and enumerating valid flat layouts from modified FAG, respectively.

For a hyper-common edge with F_h faces, it needs to calculate $F_h - 1$ angles between two faces and sort these angles to get the sequence. Therefore, by using quick sort (Shaffer, 1997; Alsuwaiyel, 1999), the total time spent in calculating the angles and sorting the sequence can be given by:

$$t = F_h - 1 + O((F_h - 1) \log(F_h - 1)) = O(F_h \log F_h)$$

In the worst case, all hyper-common edges are shared by F_h faces. Therefore, for E_h

hyper-common edges,

$$T_h = O(E_h F_h \log F_h)$$

In order to obtain all the overlapping paths with distance no more than 2 (i.e. to search for inadmissible links and exclusive link sets), each face with all its adjacent faces need to be flattened together onto a 2D plane for the purpose of detecting overlapping. The time T_e is related to how many times a face flattening operation takes place, the time cost of flattening a face and the time cost of one overlapping detection. Since the two faces represented by the two nodes of each link in the FAG will be flattened with respect to each other, the total number of face flattening operations will therefore be 2 times the number of links in the FAG. Each flattening operation needs $O(V_f)$ time (see Chapter 4). For unfolded flat patterns associated with each face, an overlapping detection will be executed. In the worst case, assuming each flat pattern contains as many edges as that in the original structure, detecting overlapping needs $O(E \log(E))$ time (see Appendix B). Therefore, T_e is:

$$T_e = 2L \times O(V_f) + F \times O(E \log(E)) = O(LV_f + FE \log(E))$$

Using the compact output method, it needs E time to construct FAG and $O(S + N + L)$ time to obtain all S spanning trees of a graph $G(N, L)$ with N nodes and L links (note that $N = F$). The time cost of constructing a flat layout and detecting overlapping remain unchanged (i.e. $O(FV_f)$ and $O(E \log(E))$, respectively) as described in Chapter 4. Assume that there are E_L exclusive link pairs. Hence detecting whether a spanning tree contains exclusive links or not needs $O(E_L)$ time. Also assume that there are S_1 spanning trees that do not contain any exclusive links. Therefore, T_l can be given by:

$$T_l = O(E + F + L + SE_L + S_1(FV_f + E \log(E)))$$

From time complexity analysis, it can be discovered that the most crucial and time costly part is T_l , where S is far larger than any of the other parameters.

CHAPTER 6

Improving Efficiency by Subdividing the FAG

As discussed in Chapter 5, by introducing a hyper-common edge validity check and topology check, layout enumeration efficiency can be greatly improved. However, for some structures, a large proportion of the total number of spanning trees can be topologically invalid and will be eliminated at the topology check stage. For example, Figure 6.1a is a structure containing some overlapping paths, Figure 6.1b is a structure containing a few hyper-common edges and Figure 6.1c contains both overlapping paths and hyper-common edges. The total number of spanning trees and number of topologically invalid spanning trees eliminated by topology check are given in Table 6.1. It can be seen from the table that even for some simple structures, the proportion of invalid spanning trees eliminated by topology check can be large (87.5% for the example Figure 6.1a). Although a topology check against a single spanning tree is relatively fast (compared to overlapping detection), there is still a need to check all spanning trees, which is quite time-consuming for a complex FAG (say 20 nodes with 40 links).

Figure 6.1. Structures with high percentage of topologically invalid spanning trees

Table 6.1. Results for the structures shown in Figure 6.1

Example	Non-overlapping Flat Layouts	Total Spanning Trees	Eliminated by Topology Recognition	Percentage of Invalid Trees
a	16	128	112	87.5%
b	96	8,847,360	8,847,264	99.999%
c	15,549	26,669,440	26,388,664	98.95%

In this chapter, attempts will be made to reduce the total number of spanning trees that need to be enumerated, by dividing the original problem into many sub-problems. This is done by sub-dividing the original FAG into a set of subgraphs (referred to as *sub-FAG*) because the sum of spanning trees of these subgraphs will be far smaller than that of the original FAG.

6.1 Subdividing FAG by Exclusive Link Sets

6.1.1 Basic Idea

For an exclusive link set $E(i, j)$ that is associated with a specific link (i, j) , all links within E cannot exist at the same time with (i, j) in a non-overlapping topologically valid spanning tree. Therefore, the corresponding FAG can be divided into two subgraphs by deleting either (i, j) or all links in E . However, there arises an important question that has to be answered in order not to miss out on any valid spanning tree. *Is there any spanning tree of the original FAG containing no exclusive links that is not a tree of either of these two subgraphs?* If the answer is positive, these subgraphs can-

not be used as a substitute for the original FAG since some topologically valid spanning trees of the original FAG are not contained within them. Fortunately, the answer to this question is negative. This is because any spanning tree that is not in either of these two subgraphs must contain (i, j) and at least one link from $E(i, j)$. Such a combination makes the spanning tree invalid because of the presence of exclusive links. From Section 5.2.1, it can be seen that the deletion of even a single link from a graph can greatly reduce the number of spanning trees. Therefore, eliminating exclusive links alternately from a FAG can result in subgraphs such that for each and every subgraph the total number of spanning trees is notably smaller and each spanning tree is topologically valid. A *sub-FAG* is defined as a connected subgraph with all the nodes of the corresponding FAG but contains no exclusive links. Let's take the non-manifold L-shape example shown in Figure 6.2 to illustrate the sub-FAG concept.

Figure 6.2. (a) non-manifold L-shape; (b) its corresponding FAG

As provided in Section 5.2.2, the exclusive link sets for each link in Figure 6.2b are:

$$\begin{aligned}
 E(1, 2) &= \{(1, 3), (1, 4), (2, 3), (2, 4)\}, & E(1, 3) &= \{(1, 4), (2, 3)\}, \\
 E(1, 4) &= \{(2, 4)\}, & E(2, 3) &= \{(2, 4)\}, \\
 E(2, 4) &= \emptyset, & E(3, 5) &= \emptyset, \\
 E(3, 6) &= \emptyset, & E(4, 5) &= \emptyset, \\
 E(4, 6) &= \emptyset, & E(5, 6) &= \emptyset.
 \end{aligned}$$

Since all links in $E(1, 2)$ are exclusive to $(1, 2)$, a subgraph G_1 can be constructed by keeping $(1, 2)$ and deleting all links in $E(1, 2)$ (as shown in Figure 6.3). However, G_1 is not a sub-FAG according to the sub-FAG definition because it is not connected.

The way to construct the second subgraph without exclusive links is to keep all elements in $E(1, 2)$ (namely $(1, 3)$, $(1, 4)$, $(2, 3)$ and $(2, 4)$) and delete the link $(1, 2)$ (shown as G_2 in Figure 6.3). However, $(1, 4)$ and $(2, 3)$ are exclusive to $(1, 3)$, so there are two further possibilities, i.e. keep $(1, 3)$ and delete $(1, 4)$ and $(2, 3)$, or keep $(1, 4)$ and $(2, 3)$ and delete $(1, 3)$. For the first possibility, it results in G_3 as shown in Figure 6.3. This is a sub-FAG since it is connected and has no exclusive links. In the second possibility, the subgraph constructed is G_4 and it contains $(1, 4)$, $(2, 3)$ and $(2, 4)$. However, $(2, 4)$ is exclusive to $(1, 4)$, hence two more subgraphs G_5 and G_6 are derived from G_4 by deleting either of these two links. It can be easily determined according to the sub-FAG definition that G_5 is a valid sub-FAG while G_6 is not. Although G_6 still contains exclusive links (i.e. $(2, 4)$ and $(2, 3)$) and can be divided into more subgraphs (shown as G_7 and G_8), the construction can actually stop at G_6 since it is no longer a connected graph. So all the succeeding subgraphs constructed by subdividing G_6 will not be connected as well, which means these subgraphs will certainly not satisfy the sub-FAG definition. Therefore, among the five subgraphs constructed without exclusive links, only G_3 and G_5 are sub-FAGs. A *sub-FAG set* S_G is defined as a collection of sub-FAGs such that for any sub-FAG $G_i = (N, L_i) \in S_G$, there is no $G_j = (N, L_j) \in S_G$ that satisfies $L_i \subset L_j$ or $L_j \subset L_i$.

6.1.2 Advantages of Subdividing FAGs

The previous section discussed the possibility of decomposing a FAG into a set of sub-FAGs by using exclusive link sets. This decomposition provides three advantages:

- avoids the topology recognition process for exclusive link sets during enumeration
- increases the probability of reducing the time taken to generate the first valid flat layout
- has a high probability of reducing the total number of spanning trees enumerated
- enables distributed computation

Figure 6.3. Constructing subgraphs with no exclusive links.

For the first advantage, since each sub-FAG contains no exclusive links, all spanning trees enumerated will surely contain no exclusive links as well. Hence no topology recognition for exclusive link sets is necessary. For the second advantage, since all spanning trees with exclusive links have been eliminated before enumeration begins, there is a higher probability of reaching the first valid flat layout faster than the method described in Chapter 5 as the latter may need more time to handle spanning trees containing exclusive links.

The third advantage is behind the original intention of decomposing the FAG. Ideally, only topologically valid spanning trees will be enumerated and be used for geometric construction of flat layouts. However, as will be discussed in Section 6.4, there is a problem of repetition among the spanning trees enumerated from the different sub-FAGs. The total number of enumerated spanning trees will thus be larger than that of topologically valid spanning trees. Nevertheless, in most cases, it will still be smaller than that of the original FAG. Therefore, the time cost in flat layout enumeration will be reduced mainly due to the reduction in number of spanning trees.

Since the FAG can be decomposed into a set of sub-FAGs, it becomes practicable to distribute the flat layout enumeration job to multiple computers, to speed up the overall procedure. This can be done by dispatching sub-FAGs to these client computers because each job on a specific sub-FAG is independent from others.

6.2 Converting Hyper-Common Edge Sequences to Exclusive Link Sets

The hyper-common edge sequence was used to eliminate those spanning trees containing invalid topologies related to hyper-common edges (see Section 5.1.3). In order to use a unified procedure for subdividing a FAG, hyper-common edge sequences have to be converted to exclusive link sets. Fortunately, the hyper-common edge sequence topology is essentially similar to the concept of exclusive link set.

Figure 6.4 shows a hyper-common edge set where the nodes are already numbered

Figure 6.4. Convert hyper-common edge sequences to exclusive link sets

in an order according to its sequence $S(e^h) = \{1, 2, 3, 4, 5, 6\}$. According to the two hyper-common edge validity criteria discussed in Section 5.1.3, namely:

1. No node along the circle is incident on more than one link.
2. No link intersect any other link within the circle.

Exclusive link sets can be easily constructed for each link in a hyper-common edge set. For example, the exclusive link set for the link $(1, 3)$ shown in Figure 6.4b is the collection of all links that violate the above two criteria if any of them exists together with link $(1, 3)$ in a spanning tree. Therefore:

$$E(1, 3) = \{(1, 2), (1, 4), (1, 5), (1, 6), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)\}$$

Moreover, for an arbitrary link (n_1, n_2) , where $n_1 < n_2$ within a hyper-common edge set, its corresponding exclusive link set can be constructed by the following equation:

$$E(n_1, n_2) = \left[\bigcup_{j \neq n_1, n_2}^N (n_1, j) \right] \cup \left[\bigcup_{j \neq n_1, n_2}^N (j, n_2) \right] \cup \left[\bigcup_{\substack{j < n_1 \text{ or } j > n_2 \\ n_1 < k < n_2}}^N (j, k) \right] \quad (6.1)$$

To avoid duplicated exclusive links among different exclusive link sets, links are removed from the exclusive link set for a specific link if they are smaller than it. Therefore Equation 6.1 is modified into:

$$E(n_1, n_2) = \left[\bigcup_{j > n_2}^N (n_1, j) \right] \cup \left[\bigcup_{j > n_1, j \neq n_2}^N (j, n_2) \right] \cup \left[\bigcup_{\substack{n_1 < j < n_2 \\ k > n_2}}^N (j, k) \right] \quad (6.2)$$

6.3 Sub-FAG Set Construction

The exclusive link sets are used to construct the sub-FAG set. The underlying principle is to alternately use exclusive links to build two subgraphs. The procedure will result in a binary tree in which nodes are valid sub-FAGs as well as disconnected subgraphs and subgraphs containing some exclusive links (as shown in Figure 6.3). Therefore, the construction of the sub-FAG set is a procedure similar to traversing nodes on this binary tree for those representing valid sub-FAGs. The root node of the binary tree (i.e. the root subgraph) is the original FAG, as shown in Figure 6.3. Each time a link is found with a non-empty exclusive link set, the current node (subgraph) will be divided into two: one is created by deleting the corresponding link; the other is created by deleting all links in its exclusive link set. The method of constructing sub-FAG set is profiled in Algorithm 6.1.

Require: Exclusive link sets are initialized.

Sort links in ascending order of their indices into a set S_L .

if no exclusive link in the FAG G_F **then**

 Output G_F to E_G and return.

end if

let l_i be the first link in S_L that $E(l_i) \neq \emptyset$.

let $G_1 = G_F \setminus l_i$.

let $G_2 = G_F \setminus E(l_i)$.

Call SearchSubFAG (G_1, S_L, i). {See Algorithm 6.2}

Call SearchSubFAG (G_2, S_L, i).

Algorithm 6.1. Construct Sub-FAG Set E_G

6.4 Repeated Spanning Trees

As discussed in Section 6.1, spanning trees for each sub-FAG contain neither exclusive links nor invalid hyper-common edge topologies. Ideally, there should be no overlapping

Require: A current subgraph G , a sorted link set S_L and a start link l_i

if G contains no exclusive links and is connected **then**

Output G to E_G and return.

else if G contains no exclusive links but is not connected **then**

Return.

end if

let l_j be the first link in S_L that is next to l_i and $E(l_j) \neq \emptyset$.

let $G_1 = G \setminus l_j$.

let $G_2 = G \setminus E(l_j)$.

Call SearchSubFAG (G_1, S_L, j).

Call SearchSubFAG (G_2, S_L, j).

Algorithm 6.2. SearchSubFAG (G, S_L, l_i)

of sub-FAGs, i.e. no two sub-FAGs will give rise to the same spanning tree. If this ideal condition always holds for all structures, sub-FAGs can be used directly to enumerate spanning trees since all these trees are topologically valid and nonrecurring. This section will analyze the problem posed by sub-FAG overlapping and how a solution can be developed to overcome it.

6.4.1 Sub-FAG Overlapping Problem

Given two graphs $G_1 = (N, L_1)$ and $G_2 = (N, L_2)$. Let $G_1 \cap G_2 = (N, L_3)$ be the graph where $L_3 = L_1 \cap L_2$. G_1 is said to be *overlapped* with G_2 here if $G_1 \cap G_2$ is a connected graph. For any two overlapping sub-FAGs (say G_i and G_j) within a sub-FAG set, all spanning trees of $G_i \cap G_j$ are shared by both G_i and G_j . Let us denote these spanning trees as *common spanning trees* of G_i and G_j . Common spanning trees will be traversed more than once if all sub-FAGs are used for spanning tree enumeration. Therefore, they may pose a problem that some spanning trees may be generated more than once.

Sub-FAG Relationships

There are three possible relations between any two sub-FAGs in a sub-FAG set.

1. G_i is a subgraph of G_j (or $G_i \subset G_j$).
2. $G_i \cap G_j$ is disconnected (i.e., they are not overlapped).
3. $G_i \cap G_j$ is connected (i.e., they are overlapped).

Case 1 is actually a subcase of case 3. Therefore, since all spanning trees in G_i are spanning trees in G_j , G_i can be directly eliminated from the sub-FAG set. It may be necessary to identify all such sub-FAGs. However, the algorithm described in Section 6.3 guarantees that case 1 will not happen in the sub-FAG sets, i.e., any sub-FAG will not be a subgraph of any other sub-FAG. To prove it, let us use a counter example. Let G_i and G_j be two different sub-FAGs in a sub-FAG set such that $G_i \subset G_j$. Moreover, let $G_j \setminus G_i$ be the subgraph within which all links are links in G_j but not in G_i . For any link $l \in (G_j \setminus G_i)$, $(G_i \cup l) \subset G_j$. Since G_j does not contain exclusive links, $G_i \cup l$ does not contain any too. Therefore, $G_i \cup l$ is also a sub-FAG. However, the algorithm requires that any deleted link must be exclusive to at least one link in G_i , i.e. l is exclusive to at least one link in G_i . Hence $G_i \cup l$ does contain exclusive links, which contradicts the previous conclusion. Therefore case 1 cannot happen in the sub-FAG set generated by the algorithm from Section 6.3.

If case 2 happens, that is, $G_i \cap G_j$ is not connected, it can be proven that G_i and G_j have no common spanning tree. This is obvious since if a spanning tree T is a common spanning tree between G_i and G_j , then T must be a spanning tree of $G_i \cap G_j$. However, $G_i \cap G_j$ is not connected, which means it has no spanning tree at all, so T is not a spanning tree as well.

For case 3, since $G_i \cap G_j$ is connected, all the spanning trees of the subgraph $G_i \cap G_j$ are common to G_i and G_j . Therefore, these spanning trees will be enumerated more than once. In order not to obtain two or more of the same flat layouts, the duplicated spanning

trees have to be eliminated. The solution for eliminating repeated spanning trees will be described in the following sections.

The Sum of Repeated Spanning Trees

Since there are repeated spanning trees between different sub-FAGs, there arises a concern that the sum of spanning trees of all sub-FAGs may be larger than that of the original FAG. If this happens, using sub-FAGs to enumerate spanning trees may be more time-consuming as compared with the methodology described in Chapter 5. On the other hand, it is hard to calculate the sum of repeated spanning trees, even between two sub-FAGs. Therefore, it may be best to discover some rules via empirical tests. Figure 6.5 shows some structures that were used to analyze this problem. Figure 6.5a and b are L-shape structures. However, for Figure 6.5a, one rectangular face at the concave corner has been removed. By erasing this face, the number of overlapping paths with distance of 2 is reduced to 1. Figure 6.5c, d and e are edge, face and corner buffers, respectively. They are generic shapes of packaging buffers used in the packaging industry.

Figure 6.5. Some structures for repeated spanning tree analysis

Table 6.2 shows the empirical results for these five examples. In the table, S_T de-

notes the sum of spanning trees enumerated from all sub-FAGs. S denotes the number of spanning trees from the original FAG. R_s is the ratio of S_T to S . If $R_s > 1$, the subdividing FAG method will result in more spanning trees and therefore the method is not preferred. In other words, the closer that R_s approaches zero, the more efficient this sub-FAG approach will be. L_I is the number of links in the original FAG that are involved in exclusive link sets. L is the number of links in the original FAG. R_I is the percentage of links that are involved in exclusive link sets.

Table 6.2. Repeated spanning tree analysis

<i>Example</i>	S_T	S	R_s	L_I	L	R_I
a	1,116	1,045	1.068	2	14	14.29%
b	4,722	8,100	0.583	5	18	27.78%
c	2,688	97,280	0.028	10	24	41.67%
d	42,288	99,456	0.425	9	36	25%
e	1,878,532	51,861,600	0.036	24	36	66.67%

From Table 6.2, it can be discovered that, in general, larger values of R_I lead to smaller values of R_s . This rule may not apply to all cases (e.g. the examples b and d) since S_T is also dependent on some other topological attributes (e.g. the locations of overlapping paths in the original FAG or the repeated involvement of a link in several overlapping paths). However, it is still an important criterion to assess the effectiveness of the subdividing FAG method for a particular structure. The rule can be formalized as: *the larger the proportion of links involved in exclusive link sets, the larger the number of spanning trees that will be omitted*. This is reasonable because having more links involved in exclusive link sets means have more links deleted from the original FAG to construct sub-FAGs. Therefore, the fewer the number of spanning trees that the newly constructed sub-FAG will contain.

6.4.2 Indexing Spanning Trees

In order to eliminate repetition, those spanning trees that have been enumerated need to be identified. One way to identify spanning trees is to index each spanning tree so that

it has a unique ‘name’ which is different from all other different spanning trees. Figure 6.6 shows the mechanism used to identify spanning trees.

Figure 6.6. Indexing spanning trees

Figure 6.6a is the FAG for the structure shown in Figure 6.2a. Each and every link of this graph can be assigned an integer (called *link index*) in running order, starting from zero. Figure 6.6b shows an array of binary bits in which each bit represents a link in the graph. The i^{th} bit represents the link indexed as i . If the i^{th} link exists in a spanning tree, the corresponding bit in the binary array will be set to 1, otherwise, it will be set to 0. Therefore, for the FAG, all bits in the binary array are 1s. Figure 6.6c shows a spanning tree of the FAG and Figure 6.6d is the corresponding binary array for this spanning tree. These binary arrays can be used as identification codes since they are unique for each spanning tree.

6.4.3 Detecting Repeated Spanning Trees

Binary arrays are used to identify different spanning trees. In order to determine whether a spanning tree has been enumerated or not, the binary arrays for all traversed spanning trees have to be recorded. The next time a new spanning tree is obtained, a check can be preformed to determine if its corresponding binary array has already been recorded. To speed up the checking of a spanning tree record, the binary arrays are stored in ascending order.

A spanning tree is said to be larger than another if its binary number (represented by the binary array) is larger than that of the other. If the binary numbers are equal, the two trees are said to be equal, i.e. the two spanning trees are the same. By applying this rule, binary arrays can be sorted and recorded sequentially.

Searching for a spanning tree in the sorted binary array is needed each time a new spanning tree is obtained from enumeration. The simplest way is to compare each entry of the sorted binary arrays with the candidate spanning tree. However, this could be time-consuming. For a sorted array with N entries, this method will need $O(N)$ time to find the right position. The efficiency can be improved by using a bisection technique. The sorted array is divided into two at its midpoint entry, with one half of smaller entries and the other of larger entries. If the midpoint entry is smaller than the candidate, the half with larger entries is further divided into two and its midpoint entry will be compared with the candidate. This procedure will continue until the candidate is found or the size of the divided half array is 1 or 0. Therefore, the time cost can be calculated by:

$$\sum_{k=1}^N \frac{1}{k} = O(\log N)$$

6.5 Overall Methodology

6.5.1 Methodology Outline

In order to apply the subdividing FAG method, the overall unfolding strategy has to be modified from that described in Chapter 5. The modified methodology is outlined in Algorithm 6.3. Several parts of the methodology is described in short since they can be found in previous algorithms. Analyzing the model for constructing exclusive link sets is described in Algorithm 5.1. The procedure for constructing sub-FAGs is described in Algorithm 6.1. And the methods to construct flat layouts and check overlapping remain unchanged as those described in Chapter 4.

Analyze model and find all exclusive link sets. {See Algorithm 5.1}

while there are more sub-FAGs **do**

Construct the next sub-FAG G_n . {See Algorithm 6.1}

while there are more spanning trees in G_n **do**

Enumerate next spanning tree T_i .

if the binary array of T_i has not been recorded **then**

Record T_i .

Use T_i to construct flat layout L_i . {See Algorithm 4.3}

if L_i is not overlapped **then**

Output L_i .

end if

end if

end while

end while

Algorithm 6.3. Subdividing FAG methodology

6.5.2 Time Complexity Analysis

Since the model analyzing phase follows the same way as in Chapter 5, the time complexity remains unchanged for this part as well. As discussed previously, most computing time is spent in the enumeration phase, i.e. enumerating spanning trees, constructing flat layouts and detecting overlapping. Therefore, reducing the number of spanning trees will shorten the computing time.

Assume that the number of spanning trees for all sub-FAGs is S_T . Although the number of enumerated spanning trees is smaller than S_T because of the repeated spanning trees, it can be assumed that, in the worst case, the number of enumerated spanning trees is S_T . Therefore, accessing the binary array records of the spanning trees costs $\log(S_T)$ time, while the complexity of flat layout constructions and overlapping detections remains as before. Hence for the enumeration phase, the time complexity is:

$$T_l = O(S_T \log(S_T)(FV_f + E \log(E)))$$

where F , V_f and E denote the number of faces, maximum number of vertices in a face and number of edges, respectively.

As compared with the methodology described in the previous chapter, the major difference in time complexity is the original S is replaced by $S_T \log(S_T)$. If $S_T \ll S$, then this modified methodology will be more efficient than the previous one.

6.5.3 Space Complexity Analysis

Previously, space complexity is not as important as time complexity. This is because after each spanning tree is used to construct a flat layout, the working memory used will be freed. However, the subdividing FAG methodology needs to record all the binary arrays for the enumerated spanning trees, and the number of these arrays can be quite large. Hence the space complexity becomes another important concern.

Each binary array has L elements, where L is the number of links in the corresponding FAG. For the worst case, the number of enumerated spanning trees can be the same as S_T . Therefore, the space complexity is $O(S_T L)$.

CHAPTER 7

Results

All of the three versions of the unfolding methodology have been implemented in a C++ program that runs on a personal computer. The program is named Layout Enumeration Tool (LET, see Appendix A). This chapter presents the flat layout design results generated from various 3D folded structure examples by the third version of the LET program (using the FAG subdividing methodology described in Chapter 6). The testing environment is:

- **Software:** Windows 2000 Professional
- **Hardware:** Intel Pentium IV 2.4GHz CPU, 512M memory and 80G hard disk.

Some of these structures are adapted from generic packaging buffer designs that can be used for cushioning products shaped like a rectangular block, while others are structures chosen or created to contain features/complications that can test the unfolding methodology. Some of these results have been published in Liu & Tai (2002) and Tai

et al. (2004a,b). Note that in order to maintain efficiency and not increase CPU time excessively, the flat layout perimeter needed to compute geometric compactness is simply obtained by summing up the lengths of the free edges of the flat layout, and hence does not take into account free edges that coincide (which in effect leads to a shorter outer boundary/perimeter). In addition, note that throughout this chapter, the total number of enumerated spanning trees listed in the result tables is calculated by summing up the numbers of spanning trees obtained from each sub-FAG. Therefore, this total number will include repeated spanning trees.

7.1 Example 1

Table 7.1 shows an example structure that is built for testing the LET program, to ensure that no invalid flat layouts will be generated. In this structure, there are five parallel hyper-common edges. By manual inspection, it can be discovered that this example has no edge unfolding. The results listed in Table 7.1 shows that the LET program did not find any non-overlapping flat layouts. In fact, it did not generate any sub-FAGs and hence there are no topologically valid spanning trees. This is due to the reason that each and every spanning tree of the original FAG contains some links that violate the topology validity criteria for hyper-common edges (see Section 5.1.3).

Table 7.1. Example 1

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	25,365
Total Sub-FAGs	0
Total Spanning Trees Enumerated	0
Total Non-overlapping Flat Layouts	0
CPU Time (s)	0.046

7.2 Example 2

The example shown in Table 7.2 does not contain any non-overlapping flat layout as well. But this time the reason is that all spanning trees contain some overlapping paths. For this example, two sub-FAGs are created based on exclusive links and 768 spanning trees are enumerated. These spanning trees contain no exclusive links, and therefore are used to construct geometric flat layouts and tested for overlapping. However, every one of them contain at least one overlapping path of distance greater than two (e.g. $\{4, 3, 2, 9\}$). Hence, no non-overlapping flat layout is found.

Table 7.2. Example 2

<div><div> </div><div>(with hidden lines)</div></div> <div><div> </div><div>(without hidden lines)</div></div>	
Total Spanning Trees of the Original FAG	97,280
Total Sub-FAGs	2
Total Spanning Trees Enumerated	768
Total Non-overlapping Flat Layouts	0
CPU Time (s)	3.296

7.3 Example 3

Table 7.3 shows a simple non-manifold L-shape structure with 6 faces. Since the structure is simple, all of its 16 non-overlapping flat layouts can be manually enumerated. Therefore, it is used to test the LET program to ensure that no valid result will be missed. The whole set of flat layouts is generated by the LET program and listed in Table 7.3. From this example, it can be seen that for a given structure, there may be more than one optimal result according to a specific optimality criterion. Moreover, there may be some results that are optimal in more than one criterion. Ideally, the best result will be the one that satisfy all optimality criteria. However, in practice, finding a flat layout that satisfies as many optimality criteria as possible is more realizable. In Table 7.3, layouts which are optimal according to any of the three optimality criteria (i.e. the most geometrically compact, the minimum overall extent and the minimum enclosing area) have been indicated as shown.

Table 7.3. Example 3

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	128
Total Sub-FAGs	2
Total Spanning Trees Enumerated	16
Total Non-overlapping Flat Layouts	16
CPU Time (s)	0.062
<i>continued on next page</i>	

continued from previous page

(a)

(b)

(c) Minimum Overall Extent

(d) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

(e)

(f) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

continued on next page

continued from previous page

(g)

(h)

(i)

(j)

(k) Minimum Overall Extent

(l) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

continued on next page

continued from previous page

(m)

(n) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

(o)

(p)

7.4 Example 4

Table 7.4 shows an example structure that is used for testing the capability of the methodology to correctly handle structures with hyper-common edge features. The interesting part of this structure is that it contains four parallel hyper-common edges. The original spanning tree enumeration method described in Chapter 4 will generate some non-overlapping flat layouts that cannot be folded (see Section 5.1.2). The topological recognition method can correctly handle these hyper-common edges, and therefore can be used to handle this structure. However, it is still based on the spanning tree enumeration. Since the number of spanning trees of the original FAG is large (8,847,360), it may need quite a long time to enumerate flat layouts by using the topological recognition method. However, when the sub-FAG method is adopted, only 118 spanning trees were enumerated and used to generate geometric flat layouts while the rest were never enumerated or tested since they all contain some exclusive links. Furthermore, note that the adjacent faces 9 and 15 are coplanar in the original 3D structure, and they are also adjacent in the flat layout (a); therefore, according to the face merging method described in the Section 4.3.2, they are combined into one face in the flat layout (a). Since all of the 96 flat layouts obtained have the same boundary shape (rectangle), only two of them are listed in Table 7.4.

Table 7.4. Example 4

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	8,847,360
Total Sub-FAGs	58
<i>continued on next page</i>	

continued from previous page																																																															
Total Spanning Trees Enumerated	118																																																														
Total Non-overlapping Flat Layouts	96																																																														
CPU Time (s)	2.171																																																														
<div><div><table><tr><td>8</td></tr><tr><td>-----</td></tr><tr><td>7</td></tr><tr><td>-----</td></tr><tr><td>6</td></tr><tr><td>-----</td></tr><tr><td>5</td></tr><tr><td>-----</td></tr><tr><td>4</td></tr><tr><td>-----</td></tr><tr><td>3</td></tr><tr><td>-----</td></tr><tr><td>2</td></tr><tr><td>-----</td></tr><tr><td>11</td></tr><tr><td>-----</td></tr><tr><td>12</td></tr><tr><td>-----</td></tr><tr><td>1</td></tr><tr><td>-----</td></tr><tr><td>10</td></tr><tr><td>-----</td></tr><tr><td>9</td></tr><tr><td>-----</td></tr><tr><td>15</td></tr><tr><td>-----</td></tr><tr><td>16</td></tr><tr><td>-----</td></tr><tr><td>13</td></tr><tr><td>-----</td></tr><tr><td>14</td></tr></table></div><div>(a)</div><div><table><tr><td>7</td></tr><tr><td>-----</td></tr><tr><td>6</td></tr><tr><td>-----</td></tr><tr><td>5</td></tr><tr><td>-----</td></tr><tr><td>4</td></tr><tr><td>-----</td></tr><tr><td>3</td></tr><tr><td>-----</td></tr><tr><td>2</td></tr><tr><td>-----</td></tr><tr><td>11</td></tr><tr><td>-----</td></tr><tr><td>12</td></tr><tr><td>-----</td></tr><tr><td>13</td></tr><tr><td>-----</td></tr><tr><td>16</td></tr><tr><td>-----</td></tr><tr><td>15</td></tr><tr><td>-----</td></tr><tr><td>14</td></tr><tr><td>-----</td></tr><tr><td>1</td></tr><tr><td>-----</td></tr><tr><td>10</td></tr><tr><td>-----</td></tr><tr><td>9</td></tr><tr><td>-----</td></tr><tr><td>8</td></tr></table></div><div>(b)</div></div>		8	-----	7	-----	6	-----	5	-----	4	-----	3	-----	2	-----	11	-----	12	-----	1	-----	10	-----	9	-----	15	-----	16	-----	13	-----	14	7	-----	6	-----	5	-----	4	-----	3	-----	2	-----	11	-----	12	-----	13	-----	16	-----	15	-----	14	-----	1	-----	10	-----	9	-----	8
8																																																															

7																																																															

6																																																															

5																																																															

4																																																															

3																																																															

2																																																															

11																																																															

12																																																															

1																																																															

10																																																															

9																																																															

15																																																															

16																																																															

13																																																															

14																																																															
7																																																															

6																																																															

5																																																															

4																																																															

3																																																															

2																																																															

11																																																															

12																																																															

13																																																															

16																																																															

15																																																															

14																																																															

1																																																															

10																																																															

9																																																															

8																																																															

7.5 Example 5

Table 7.5 shows a structure used for testing face merging problems and hyper-common edge problems. Face 2 and 3, face 5 and 6, and face 8 and 9 are coplanar, respectively. Therefore, if any of these face pairs is coplanar in a flat layout, it can be merged into a single face. This structure also contains 3 unparallel connected hyper-common edges ($\{2, 3, 7\}$, $\{5, 6, 7\}$ and $\{8, 9, 7\}$), so it can also be used as a test case for the unparallel connected hyper-common edges. The results show that the number of spanning trees that need to be enumerated is reduced by 63%. Face 2 and 3, 5 and 6, and 8 and 9 are merged respectively whenever any of the three pairs is connected in a flat layout. The first three flat layout (a), (b) and (c) are the optimal results according to the three optimality criteria, while the remaining five layouts (d) to (h) are randomly selected sample results from the total of 1,183 layouts generated.

Table 7.5. Example 5

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	14,175
Total Sub-FAGs	24
Total Spanning Trees Enumerated	5,184
Total Non-overlapping Flat Layouts	1,183
CPU Time (s)	13.078
<i>continued on next page</i>	

continued from previous page

(a) Most Geometrically Compact

(b) Minimum Overall Extent

(c) Minimum Enclosing Area

(d)

(e)

(f)

continued on next page

continued from previous page

(g)

(h)

7.6 Example 6

This structure contains a through hole (the through hole is formed by faces 10, 11, 12 and 13, see Table 7.6). The interesting feature of this structure is that one end of the through hole is connected to the inner ring of face 1 while the other end is formed by the edges of a few faces (i.e. face 6, 7, 8 and 9). The structure can be unfolded without any modification (i.e. there is no need to split face 1 into two faces). As shown by the results data listed in Table 7.6, only about 11% of the original spanning trees are enumerated. This is due to the reason that the corresponding FAG contains some inadmissible links as well (the inadmissible links are $\{1, 10\}$, $\{1, 11\}$, $\{1, 12\}$ and $\{1, 13\}$).

Table 7.6. Example 6

<div><div> <p>(with hidden lines)</p></div><div> <p>(without hidden lines)</p></div></div>	
Total Spanning Trees of the Original FAG	2,558,976
Total Sub-FAGs	1
Total Spanning Trees Enumerated	284,089
Total Non-overlapping Flat Layouts	23,661
CPU Time (s)	1033.596
continued on next page	

continued from previous page

(a) Most Geometrically Compact

(b) Minimum Overall Extent

(c) Minimum Enclosing Area

(d)

(e)

(f)

continued on next page

continued from previous page

(g)

(h)

7.7 Example 7

The example structure shown in Table 7.7 also contains a through hole. Although the two sides of the through hole are connected to the respective inner rings of two different faces (i.e. face 1 and 6), this structure has edge unfoldings without the need for any manual modifications. From the flat layouts listed in Table 7.7, it can be found that any of the two opposite faces that form the through hole (i.e. face 7 and 9, or face 8 and 10) can be unfolded into either of the inner rings, respectively. Therefore, the two face pairs can be unfolded to the two inner rings. This structure contains no inadmissible link; however, it contains exclusive links. Two sub-FAGs are constructed from these exclusive links and by sub-dividing the original FAG, only about 0.5% of the original spanning trees are generated and all of these 0.5% spanning trees lead to non-overlapping flat layouts.

Table 7.7. Example 7

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	147,456
Total Sub-FAGs	2
Total Spanning Trees Enumerated	768
Total Non-overlapping Flat Layouts	768
CPU Time (s)	5.015
<i>continued on next page</i>	

continued from previous page

(a) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

(b) Minimum Overall Extent

(c) Most Geometrically Compact,
Minimum Overall Extent
& Minimum Enclosing Area

(d)

(e)

(f)

continued on next page

continued from previous page

(g)

(h)

7.8 Example 8

The example structure shown in Table 7.8 is used to illustrate the treatment of inadmissible links. There are a total of six inadmissible links in this structure (i.e. $\{2, 3\}$, $\{2, 9\}$, $\{2, 10\}$, $\{4, 3\}$, $\{4, 9\}$ and $\{4, 10\}$). When the program removes these inadmissible links, the number of spanning trees is greatly reduced. Only about 3.3% of the original spanning trees are enumerated and used for constructing geometric flat layouts. So from this example, it can be seen that the deletion of inadmissible links has a great impact on simplifying the overall flat layout enumeration problem.

Table 7.8. Example 8

<div><div> </div><div>(with hidden lines)</div></div> <div><div> </div><div>(without hidden lines)</div></div>	
Total Spanning Trees of the Original FAG	150,528
Total Sub-FAGs	1
Total Spanning Trees Enumerated	4,904
Total Non-overlapping Flat Layouts	2,632
CPU Time (s)	21.265
<div><div> </div><div>(a) Most Geometrically Compact</div><div> </div><div>(b) Minimum Overall Extent</div></div>	
continued on next page	

continued from previous page

(c) Minimum Enclosing Area

(d)

(e)

(f)

(g)

(h)

7.9 Example 9

The example structure shown in Table 7.9 can be used as a face buffer to protect a rectangular face of a product within the packaging. There is a depression at the top of the structure and one side (face) of the product can fit into this depression. Face 9 acts as a rib that makes the structure more robust. In order to ensure the unfolding is possible, the top face is split into two (faces 5 and 6) by two extra edges (the two edges connecting the outer loop and the inner ring of the top face). The first three flat layouts in Table 7.9 are the optimal results according to the three optimality criteria while the remaining five layouts are randomly selected sample results from the total of 3,971 layouts generated.

Table 7.9. Example 9

(with hidden lines)	(without hidden lines)
Total Spanning Trees of the Original FAG	99,456
Total Sub-FAGs	28
Total Spanning Trees Enumerated	78,060
Total Non-overlapping Flat Layouts	3,971
CPU Time (s)	123.875

(a) Most Geometrically Compact	(b) Minimum Overall Extent
continued on next page	

continued from previous page

(c) Minimum Enclosing Area

(d)

(e)

(f)

(g)

(h)

7.10 Example 10

The example structure shown in Table 7.10 can be used as an edge buffer to protect an edge of a product. This structure has the same FAG (i.e. the same topology) as Example 2. However, since there are differences in vertex coordinates, the shape and flat layout enumeration results are very different from those in Example 2. Eight sub-FAGs are constructed and 448 non-overlapping flat layouts are obtained (as a comparison, two sub-FAGs were constructed and no non-overlapping flat layout was found for Example 2). Moreover, faces 1 and 4 share more than one common edge, therefore this example can be used to verify that the problem described in Section 4.1.2 does not pose any difficulties to the program because the results obtained show that no invalid layouts are generated due to this problem of sharing more than one common edge. The flat layout (c) is optimal both in overall extent and enclosing area criteria. Therefore, this result may be preferable to layout (b) which is optimal only in its overall extent.

Table 7.10. Example 10

<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	97,280
Total Sub-FAGs	8
Total Spanning Trees Enumerated	3,072
Total Non-overlapping Flat Layouts	448
CPU Time (s)	15.375
<i>continued on next page</i>	

continued from previous page

(a) Most Geometrically Compact

(b) Minimum Overall Extent

(c) Minimum Enclosing Area
& Minimum Overall Extent

(d)

(e)

(f)

(g)

(h)

7.11 Example 11

The structure shown in Table 7.11 can be used as a packaging cushion for protecting the rectangular corner of a product (one corner of the product can sit in the corner formed by face 1, 2 and 3). The irregular faces of the structure are intentionally made so as to enable the faces in the results to be more easily identified. From the test data, only about 59% of the original spanning trees are enumerated. And among these enumerated spanning trees, only about 9.8% can lead to non-overlapping flat layouts (while others are either repeated spanning trees or contain some overlapping paths with distance greater than 2).

Table 7.11. Example 11

<div><div> </div><div>(with hidden lines)</div></div> <div><div> </div><div>(without hidden lines)</div></div>	
Total Spanning Trees of the Original FAG	676,818
Total Sub-FAGs	18
Total Spanning Trees Enumerated	400,154
Total Non-overlapping Flat Layouts	39,029
CPU Time (s)	831.843
<div><div> </div><div>(a) Most Geometrically Compact</div></div> <div><div> </div><div>(b) Minimum Overall Extent</div></div>	
continued on next page	

continued from previous page

(c) Minimum Enclosing Area

(d)

(e)

(f)

(g)

(h)

7.12 Example 12

The structure shown in Table 7.12 is another example that can be used as an edge buffer. The difference between this example and Example 10 is that in this structure there are two outer ribs which help to improve the impact absorption/cushioning effect. The number of faces in this structure is six more than that of Example 10 but the original number of spanning trees is almost 7,000 times more. This shows that slight changes in the structure configuration may dramatically increase/decrease the problem complexity. Due to the FAG subdividing, only about 0.1% of the original spanning trees need to be enumerated.

Table 7.12. Example 12

<div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> <p>(with hidden lines)</p> </div> <div style="text-align: center;"> <p>(without hidden lines)</p> </div> </div>	
Total Spanning Trees of the Original FAG	675,584,000
Total Sub-FAGs	79
Total Spanning Trees Enumerated	1,010,160
Total Non-overlapping Flat Layouts	76,096
CPU Time (s)	7590.921
<div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> <p>(a) Most Geometrically Compact</p> </div> <div style="text-align: center;"> <p>(b) Minimum Overall Extent</p> </div> </div>	
<i>continued on next page</i>	

continued from previous page

(c) Minimum Enclosing Area

(d)

(e)

(f)

(g)

(h)

7.13 Example 13

The structure shown in Table 7.13 can be used as a partition which can protect as well as contain a modular or multi-component product. The main product (e.g. a digital camera, a telephone, etc.) can be put in the depression, while the product accessories (e.g. an adapter or cables, etc.) can be put in the side compartment. This structure features several complications addressed in this work, e.g. hyper-common edges, face-merging and non-manifold complexities. From the result, it can be seen that only 3.7% of the original spanning trees are enumerated and used for constructing flat layouts.

Table 7.13. Example 13

<div><div> </div><div>(with hidden lines)</div></div> <div><div> </div><div>(without hidden lines)</div></div>	
Total Spanning Trees of the Original FAG	14,694,375
Total Sub-FAGs	431
Total Spanning Trees Enumerated	550,708
Total Non-overlapping Flat Layouts	33,171
CPU Time (s)	982.921
<div><div> </div><div>(a) Most Geometrically Compact</div></div> <div><div> </div><div>(b) Minimum Overall Extent</div></div>	
continued on next page	

continued from previous page

(c) Minimum Enclosing Area

(d)

(e)

(f)

(g)

(h)

7.14 Comparison of the Three Versions

All of the above examples have been run on the three versions of the LET program. Table 7.14 shows a comparison of the CPU time consumed by the different versions. Note that for Example 4, since no hyper-common edge validity detection is applied in version 1, some of the results cannot be folded. Therefore, we did not test this example on version 1. From the table, it can be seen that the efficiencies of versions 2 and 3 are much higher than that of version 1. Between versions 2 and 3, their performance is quite close (because topological recognition for exclusive links and hyper-common edges are also efficient). However, for Examples 1, 4, 12 and 13, subdividing FAG method (version 3) is significantly more efficient than the topological recognition method (version 2). This is because as more links are involved in exclusive links and hyper-common edge sets, more sub-FAGs will be constructed and more topologically invalid spanning trees will not be enumerated. Note that version 1 is not used to test the last two examples, so there is no test result for version 1. Also note that if distributed computing is enabled for subdividing FAG method, the performance can be expected to be improved.

Table 7.14. CPU Time Comparison of the Three Versions of LET

Example	Version 1(s)	Version 2(s)	Version 3(s)
1	84	1.015	0.046
2	255.5	2.313	3.296
3	0.25	0.078	0.062
4	N.A.	365.891	2.171
5	32.5	9.203	13.078
6	8062.469	1044.172	1033.109
7	366.609	9.047	5.015
8	364.985	21.688	21.265
9	356.469	62.219	123.875
10	275.844	13.938	15.375
11	2177.11	633.375	831.843
12	N.A.	9913.36	7590.921
13	N.A.	1307.75	982.921

CHAPTER 8

Conclusion and Recommendations

8.1 Conclusion

A basic unfolding methodology has been developed for enumerating all non-overlapping flat layouts for a given 3D folded structure comprising piece-wise flat surfaces. The flat layouts can be then folded along some creases into the original 3D structure. The topology information of the folded structure is represented by a Face Adjacency Graph (FAG) and the topology of the unfolded flat layout is a spanning tree of the graph. Therefore, all necessary spanning trees of the FAG are enumerated. By using the face connectivity information provided by each spanning tree, the 3D structure is unfolded into its 2D flat layouts, and those with no overlapping faces are valid results.

The methodology was then further developed to handle greater complexities within structures (such as the hyper-common edges found in non-manifold structures). The implications of the hyper-common edge sequence were studied and implemented as a topology recognition of the validity of the unfolding and hence the validity of any given

spanning tree. This methodology improvement was based on the idea of making the most out of the topology information to detect invalid spanning trees and so avoid constructing invalid geometric flat layouts and having to check for overlapping. The concept of overlapping paths was also proposed for this purpose. Each overlapping path is a path in the FAG that will surely lead to overlapped flat layouts if it is present in a spanning tree. Since the number of overlapping paths can be too many to be efficiently counted, only those with distance no more than 2 were identified and used. The link involved in a 1-distance overlapping path was named an ‘inadmissible link’ whilst any two links involved in a 2-distance overlapping path were named ‘exclusive links’. In the modified methodology, inadmissible links and exclusive links are searched and recorded before flat layout enumeration begins. Spanning trees are checked against these topologies and only those without any inadmissible links or exclusive links are further processed.

For the reason that the corresponding flat layout will be overlapped if any two exclusive links are present in a spanning tree, the unfolding methodology was further enhanced by subdividing the FAG into a set of subgraphs (which were named ‘sub-FAGs’). Each sub-FAG contains no exclusive links (or inadmissible links), and therefore, each spanning tree of a sub-FAG is valid and can be used directly to construct a geometric flat layout and checked for overlapping (in comparison, the preceding methodology will have to check the existence of inadmissible links and exclusive links against each spanning tree enumerated). Moreover, since the flat layout enumeration from each sub-FAG is independent from others, distributed computing can be utilized for implementing this methodology to improve the overall speed.

Some other issues related to unfolding methodology (e.g. treatment for depressions, protrusions and through-holes, face merging problems and optimality criteria) have also been discussed in this work.

8.2 Recommendations

In this work, the main contributions towards an effective unfolding methodology have been, firstly, the treatment of complexities such as hyper-common edges in non-manifold structures. Secondly, a topological recognition method based on the concept of overlapping path was proposed to improve the overall efficiency. Thirdly, a FAG subdividing methodology was developed to remove the foreknown invalid results at the very beginning and to divide the problem into a group of sub-problems (which can be tackled by distributed computing).

As for future promising directions of research, efforts and improvements are recommended to be put into the areas discussed in the following sections.

8.2.1 Efficiency Improvement

In the current research, only those overlapping paths with distance less than or equal to 2 are used for improving methodology efficiency. However, as a suggestion, overlapping paths with other greater distances can also be used for topological recognition during the enumeration process. It is not necessary to identify all overlapping paths at the very beginning (since the number of overlapping paths can be even more than that of spanning trees). Instead, overlapping paths (of distances greater than 2) can be identified dynamically during the flat layout enumeration procedure itself. The newly identified overlapping paths can be added to an overlapping path library (in order not to slow down the enumeration, the size of the library should not be too large). Next time a spanning tree is found, overlapping paths in the library can be tested for first before constructing the corresponding flat layout. Alternatively, if a way can be found to smoothly integrate this dynamic overlapping path recognition with the FAG subdivision technique, the overall efficiency may be improved even further.

Another way of improving efficiency, as mentioned previously, is to distribute sub-FAGs to different computers. But this still has complexities. For example, some sub-FAGs only contain a few spanning trees which can be enumerated within seconds, while

some other contain fairly large number of spanning trees. Therefore, how the sub-FAGs are to be distributed remains as another challenging problem.

8.2.2 Handling of Unmanageable Structures

Currently, the methodology is able to generate all non-overlapping flat layouts for a given 3D structure, provided that the structure contains at least one edge unfolding. However, for those structures that contain no edge unfolding, the methodology will generate no result, unless the structure is modified by the designer manually. The modification rule for structures containing depressions, protrusions and through holes has already been provided in Section 4.3.1. For future improvements, the structure modification may be automated by implementing an algorithm that detects those structures with no edge unfolding and then exercises the modification rules.

On the other hand, it still remains unstudied for some other problematic structures. For example, Figure 8.1a shows such a problematic structure. The structure contains no edge unfoldings, so our methodology is not capable of generating any result. One possible solution to this structure is to duplicate the two central vertical faces. The duplication of these two faces can lead to an unfolding shown in Figure 8.1b. More studies need to be done to develop more systematic ways to manage such structures.

Figure 8.1. An unmanageable structure

8.2.3 Intersecting Faces

An important and widely used technique in the packaging industry is to cut slots in intersecting faces to simplify structures. For example, the central partitioning portion of the structure shown in Figure 8.2a can be extracted out. So the original structure is divided into two component: one is a open box while the other is formed by three intersecting faces. The intersecting face component do not have a direct unfolding. Anyway, by separating this component into three faces and cutting slots in them (as shown in Figure 8.2), the separated faces can be put back together to form the component. This technique can greatly simplify the structure, which will in turn greatly simplify the layout design task (and possible folding processes). However, in the previous work, the main objective is to enumerate all possible one-piece layouts. Therefore, this situation was not taken into consideration. How to recognize the features inside a given structure that can be dealt with by this technique is a difficult but very interesting computational problem. Feature definition and recognition technology need to be developed before this cutting slot method can be automated.

Figure 8.2. Intersecting Faces

8.2.4 Design for Folding

Currently, the generated flat layouts are only schematic. Thickness of materials and securing of the folded 3D structures are not taken into consideration. However, for real applications, these factors have to be considered.

Figure 8.3a shows a very simple structure with material thickness considered. Figure 8.3b is an applicable flat layout for this structure that can be manufactured. Schematic faces in the flat layout are shifted away from each other, to give allowance for material thickness. Flaps are also added to the flat layout for holding and securing the folded structure in place and forming a robust 3D structure. In the future, an algorithm for automatically assigning flaps and adjusting flat layouts for material thickness is desired.

Figure 8.3. Addition of flaps and allowance for material thickness

APPENDIX **A**

Layout Enumeration Tool (LET)

A.1 Version 1 System Description

At the system design point of view, the whole procedure of unfolding a 3D folded structure can be looked upon as a flat layout generation system, where the inputs are 3D folded structures and outputs are graphical flat patterns displayed on computer screens and flat pattern description files which can be stored permanently. The system diagram shown in Figure A.1 illustrates the implementation of the methodology described in Chapter 4. The system consists of three subsystems, namely I/O subsystem, preprocessing subsystem and layout enumeration subsystem, which will be described separately.

A.1.1 I/O Subsystem

I/O subsystem is in charge of exchanging all necessary information with files or devices outside the system.

Model I/O Model I/O reads/writes 3D structure description files from/to storage de-

Figure A.1. LET System: version 1

vices like floppies and hard disks, etc.

Graphics I/O Graphics input and output operations include outputting 3D structures and layouts to the computer screens and handling user graphics operations, such as zoom, rotate, translate scene and select objects.

Layout I/O Layout I/O controls inputting/outputting layouts from/to storage devices.

A.1.2 Preprocessing Subsystem

The preprocessing subsystem is developed for making preliminary preparations for the flat layout enumeration. The only preliminary preparation for the version 1 is to construct the FAG from geometric connectivities between the faces of a given 3D structure. The FAG constructed by preprocessing subsystem is an important topology entity that will be used repeatedly in a flat layout enumeration procedure.

A.1.3 Flat Layout Enumeration Subsystem

It is the flat layout enumeration subsystem that searches for all possible valid flat layouts. It uses the FAG constructed by the preprocessing subsystem to enumerate all spanning trees. Each spanning tree is subsequently used to unfold the 3D structure geometrically into a flat layout. This flat layout may contain some overlapped faces, therefore, overlapping detection is performed on this layout. Only valid flat layouts will be pushed to entity pool and in turn saved to flat layout description files.

A.1.4 Entity Pool

Entity pool in the LET system is the collection of all geometric entities like vertices, edges, loops and faces, as well as all topological entities like hyper-common edge sequences, exclusive link sets and FAGs, etc. All subsystems interact with the entity pool to read/write information related to the 3D structure.

A.2 Version 2 System Description

Figure A.2 illustrates the implementation of the methodology described in Chapter 5. As compared with version 1, the I/O subsystem remains unchanged. Therefore, only the changed subsystems are explained in this section.

A.2.1 Preprocessing Subsystem

In version 2, more topological information is gathered by the preprocessing subsystem. Hyper common edge sequences and exclusive link sets are initialized and stored in the entity pool. These topological information provides the system the ability of performing topology check on spanning trees, to speed up the overall flat layout enumeration procedure.

A.2.2 Flat Layout Enumeration Subsystem

The difference in the flat layout enumeration subsystem between version 1 and 2 is that in version 2, an additional process to check the topology is integrated into the system.

Figure A.2. LET System: version 2

This process checks the topological validity (i.e. hyper-common edge validity and overlapping paths identified as exclusive links) on each spanning tree. It will identify all invalid spanning trees and discard them.

A.3 Version 3 System Description

LET version 3 is the implementation of the methodology described in Chapter 6. If there exist some exclusive links in the FAG for a specific 3D structure, this FAG will be decomposed into a set of sub-FAGs such that each spanning tree of any sub-FAG will be topologically valid. Figure A.3 illustrates the version 3 LET system.

A.3.1 Preprocessing Subsystem

Although the preprocessing subsystem in version 3 is quite similar to that of version 2, there is a slight difference. In version 3, hyper-common edge sequences are converted to exclusive link sets and are dispatched to the FAG links. The rationale for this conversion

Figure A.3. LET System: version 3

is to have a unified process to handle the FAG decomposing problem.

A.3.2 Flat Layout Enumeration Subsystem

To enumerate flat layouts, the FAG is firstly decomposed into a set of sub-FAGs. This task is actually achieved by a process to construct the sub-FAG set. Each sub-FAG is then used to enumerate spanning trees and those having been traversed before will be skipped by a repeated spanning tree detection process, while the rest will be directly used to generate flat layouts.

A.3.3 Main User Interface Screenshot

Figure A.4 shows the main user interface of LET version 3.

Figure A.4. LET main user interface screenshot

APPENDIX B

Computational Geometry for Overlapping Detection

Some techniques from computational geometry are used to implement overlapping detection algorithm between two simple polygonal faces. One technique is point location search; the other is intersection detection within a group of line segments.

B.1 Point Location Search

Given a point p and a simple polygon Q , a way of identifying the relative position of p against Q , i.e. to test whether p lies inside Q , is presented in Akl & Lyons (1993).

An example is given in Figure B.1 to illustrate the method of point location query.

Let $P = \{p_1, p_2, p_3\}$ be a group of points whose locations are to be queried. Firstly, draw a vertical line through one point of P , say, p_i ($i = 1, 2, 3$) and find the intersection points between this line and the edges of Q : If the number of such points above p_i is odd, then p_i is inside Q . For example, the numbers of such points above p_1 , p_2 and p_3 are 2, 2 and 1, respectively, thus we can conclude that p_1 and p_2 are outside Q while p_3

Figure B.1. Point location query

is inside.

There are three cases for the point location query problems that need special attention. The point p_4 locates on the boundary of the polygon shown in Figure B.1. For this case, the point is considered as located inside the polygon. The second special case is the situation when the vertical line drawn from an querying point (e.g. p_5 in Figure B.1) passes one or more vertices of the polygon, rather than edges. If the two edges that connecting with an intersecting vertex lie on the same side of the vertical line, the number of intersection times at this vertex will be counted as two; otherwise, the number of intersection times at this vertex will be counted as one. The third special case is illustrated in Figure B.1 point p_6 . The vertical line overlaps one or more edges of the polygon. The treatment of this case is similar to the previous one. If the two edges that connecting with the overlapped edge lie on the same side of the vertical line, the number of intersection times will increase by two; otherwise, the number of intersection times will increase by one.

B.2 Intersection Detection Within A Group of Line Segments

Intersection detection within different kinds of geometric entities is a classic research field in computational geometry (Mount, 1992, Möller, 1997, Atallah & Goodrich, 1986, Szirmay-Kalos & Márton, 1998 and Žalik, 2001). In order to eliminate overlapped un-

foldings, a way of detecting overlapping among faces should be developed. The method applied here is based on detecting the intersection within a group of line segments. Numerous efficient algorithms for detecting intersections among line segments were developed (Atallah & Goodrich, 1986 and Žalik, 2001). Here a well-known method called sweep-line technique (Žalik, 2001) is used.

An example is given in Figure B.2 to show the way an intersection of two faces is detected.

Figure B.2. Search for intersections using sweep-line technique

F_1 and F_2 are two faces with no rings, i.e. inner loops within faces. The edges of F_1 and F_2 are sorted into an array (from left to right) according to the smaller x -coordinate of the two vertices bounding them. The result from our example in Figure B.2 is shown in Table B.1.

Table B.1. Sorted edges

e_{11}	e_{17}	e_{18}	e_{12}	e_{13}	e_{16}	e_{21}	e_{26}	e_{25}	e_{22}	e_{14}	e_{24}	e_{15}	e_{23}
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

According to Žalik (2001), s is a sweep-line that moves in the x -direction. Firstly, it (s_0) is moved to first edge e_{11} . This edge can be called “*placed edge*”. Then we search for a candidate that may intersect the placed edge. A *candidate edge* means that

the minimum x -coordinate of the edge must be smaller or equal to the maximum x -coordinate of the placed edge where the sweep line is positioned. An edge of the same face as the placed edge cannot be a candidate edge, because the face is simple, namely, the face will not intersect itself. Moreover, the index of a candidate edge within the sorted edge array should be greater than that of the placed edge. For example, when e_{14} is the placed edge, then e_{23} and e_{24} can be candidate edges for this placed edge. After all candidates for a placed edge are identified, we shall move to the next placed edge and repeat the sweep-line procedure until the placed edge is the last edge.

Once a candidate edge is found, we must detect the intersections between the placed edge and the candidate. Let us suppose the sweep-line position is at s_6 . So the placed edge is e_{14} , and the candidates are e_{23} and e_{24} .

Figure B.3. Intersection detection between two line segments

Let us label the two vertices of e_{14} as a_{14} and b_{14} , and those of e_{24} as a_{24} and b_{24} . Draw two vertical lines at the x -coordinates of the vertices, of which the x -coordinates are neither maximum nor minimum (l_1, l_2 in Figure B.3) among these four vertices. Obviously in this case, l_1 and l_2 will intersect both e_{14} and e_{24} . Let cy_1 be the y increment between the two intersections of l_1, e_{14} and l_1, e_{24} . Similarly, let cy_2 be the y increment between the two intersections of l_2, e_{14} and l_2, e_{24} . If cy_1 and cy_2 both equal to zero, it means two lines are partly or wholly overlapped. If one of cy_1 and cy_2 equals to zero,

APPENDIX B. COMPUTATIONAL GEOMETRY FOR OVERLAPPING DETECTION 157

it means one vertex of an edge lies on the other edge. If the product of cy_1 and cy_2 is greater than zero, it means e_{14} and e_{24} do not intersect; if the product of cy_1 and cy_2 is smaller than zero, they intersect (i.e. the two faces overlap).

According to the previous studies (Žalik, 2001 and Atallah & Goodrich, 1986), detecting intersections within n line segments by using sweep-line technique needs $O(n \log(n))$ time.

References

- Agarwal, P. K., Aronov, B., O'Rourke, J., & Schevon, C. A. (1997). Star unfolding of a polytope with application. *SIAM J. Comput.*, 26, 1689–1713.
- Akl, S. G. & Lyons, K. A. (1993). *Parallel Computational Geometry*. Englewood Cliffs, NJ: Prentice Hall.
- Alsuwaiyel, M. (1999). *Algorithms: Design Techniques and Analysis*. Singapore: World Scientific.
- Ansaldi, S., Floriani, L. D., & Falcidieno, B. (1985). Geometric modeling of solid objects by using a face adjacency graph representation. *SIGGRAPH*, 19(3), 131–139.
- Atallah, M. J. & Goodrich, M. T. (1986). Efficient plane sweeping in parallel (preliminary version). In *Proceedings of the 2nd Annual ACM Symposium on Computational Geometry*, (pp. 216–225)., Yorktown Heights, New York, USA.
- Ballard, D. H. & Brown, C. M. (1982). *Computer Vision*. USA: Prentice-Hall.
- Bangay, S. (2000). From virtual to physical reality with paper folding. *Computational Geometry*, 15, 161 – 174.
- Bern, M., Demaine, E. D., Eppstein, D., & Kuo, E. (1999). Ununfoldable polyhedra. In *Proceedings of 11th Canadian Conference on Computational Geometry*, (pp. 13–16)., Vancouver, British Columbia, Canada.
- Bern, M., Demaine, E. D., Eppstein, D., Kuo, E., Mantler, A., & Snoeyink, J. (1999). Ununfoldable polyhedra with triangular faces. In *Proceedings of 4th Workshop on Computational Geometry*, (pp. 15–16)., Baltimore, Maryland.

- Bern, M. & Hayes, B. (1996). The complexity of flat origami. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, (pp. 175–183)., Atlanta, Georgia, United States.
- Biedl, T., Demaine, E., Demaine, M., Lubiw, A., Overmars, M., O'Rourke, J., Robbins, S., & Whitesides, S. (1998). Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, (pp. 70–71).
- Boissonnat, J. D. (1984). Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4), 266–286.
- Brody, A. L. & Marsh, K. S. (Eds.). (1997). *The Wiley encyclopedia of packaging technology* (2nd ed.). New York: John Wiley.
- Buekenhout, F. & Parker, M. (1998). The number of nets of the regular convex polytopes in dimension ≤ 4 . *Discrete Math.*, 186(1-3), 69–94.
- Bush, R. & Squin, C. (1999). Synthesis of bent sheet metal parts from design features. In *Proceedings of 5th Symposium on Solid Modeling and Applications*, (pp. 119–129)., Ann Arbor MI.
- Cetin, O. L. & Saitou, K. (2004a). Decomposition-based assembly synthesis for maximum structural strength and modularity. *Journal of Mechanical Design*, 126, 244–253.
- Cetin, O. L. & Saitou, K. (2004b). Decomposition-based assembly synthesis for structural modularity. *Journal of Mechanical Design*, 126, 234–243.
- Chatani, M. (1984). *Pop-up origamic architecture*. Tokyo: Ondorisha Publishers.
- Chatani, M. (1986). *Pop-up greating cards*. Tokyo: Ondorisha Publishers.

- Chuang, S. H. & Henderson, M. R. (1994). Using subgraph isomorphisms to recognize and decompose boundary representation features. *Journal of Mechanical Design*, 116, 793–800.
- Chuang, S. H. & Huang, S. F. (1996). Feature decomposition from solid models for automatic flattening. *Computer-Aided Design*, 28(6/7), 473–481.
- Dai, J. S. & Jones, J. R. (1999). Mobility in metamorphic mechanisms of foldable/erectable kinds. *Journal of Mechanical Design*, 121, 375–382.
- Dai, J. S. & Jones, J. R. (2002). Kinematics and mobility analysis of carton folds in packing manipulation based on the mechanism equivalent. In *Proceedings of Institution of Mechanical Engineers*, volume 216, Part C, (pp. 959–970).
- Demaine, E., Demaine, M., Lubiw, A., O'Rourke, J., & Pashchenko, I. (1999). Metamorphosis of the cube. In *8th Annual Video Review of Computational Geometry and Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, (pp. 409–410)., Miami Beach, Florida.
- Demaine, E. D., Demaine, M. L., Lubiw, A., & O'Rourke, J. (2000). Examples, counterexamples, and enumeration results for foldings and unfoldings between polygons and polytopes. Technical Report 069, Smith College, Northampton, MA. LANL ArXive cs.CG/0007019.
- Demaine, E. D., Demaine, M. L., Lubiw, A., & O'Rourke, J. (2002). Enumerating foldings and unfoldings between polygons and polytopes. *Graphs and Combinatorics*, 18(1), 93–104.
- Demaine, E. D., Demaine, M. L., & Mitchell, J. S. (2000). Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. *Computational Geometry: Theory and Applications*, 16, 3–21.
- Deo, N. (1974). *Graph theory with their applications to engineering and computer science*. USA: Prentice-Hall.

- Even, S. (1979). *Graph Algorithms*. London: Pitman.
- Fu, D. H. (1993). *Unfolding geometries: A utility for manufacturing applications and engineering analysis*. Ph.d. thesis, Worcester Polytechnic Institute.
- Fu, M. W., Ong, S. K., Lu, W. F., Lee, I. B. H., & Nee, A. Y. C. (2003). An approach to identify design and manufacturing features from a data exchanged part model. *Computer Aided Design*, 35(11), 979–993(15).
- Fukuda, K. (1997). *Strange unfoldings of convex polytopes*. http://www.ifor.math.ethz.ch/~fukuda/unfold_home/unfold_open.html, September, 2001.
- Gabow, H. N. & Myers, E. W. (1978). Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, 7(3), 280–287.
- Gadh, R. & Prinz, F. B. (1995). Automatic determination of feature interactions in design-for-manufacturing analysis. *Journal of Mechanical Design*, 117, 2–9.
- Holland, P., Standring, P. M., Long, H., & Mynors, D. J. (2002). Feature extraction from step (iso 10303) cad drawing files for metalforming process selection in an integrated design system. *Journal of Materials Processing Technology* 125-126, 446–455.
- Hull, T. (1994). On the mathematics of flat origamis. *Congressus Numerantium*, 100, 215–224.
- Hull, T. (1996). A note on "impossible" paper folding. *American Mathematical Monthly*, 103(3), 242–243.
- Hull, T. (1997). Planar graphs and modular origami. In *Origami Science and Art: Proceedings of the Second International Meeting of Origami Science and Scientific Origami*, (pp. 151–159). Koryo Miura ed., Seian University of Art and Design, Otsu, Japan.

- Jagirdar, R., Jain, V. K., & Batra, J. L. (2001). Characterization and identification of forming features for 3-d sheet metal components. *International Journal of Machine Tools & Manufacture*, 41, 1295–1322.
- Ji, Q. & Marefat, M. M. (2003). A dempster-shafer approach for recognizing machine features from cad models. *Pattern Recognition* 36, 1355–1368.
- Kapoor, S. & Ramesh, H. (1992). Algorithms for enumerating all spanning-trees of undirected and weighted graphs. In *Proceedings 2nd Workshop Algorithms & Data Structures*, (pp. 461–472).
- Kapoor, S. & Ramesh, H. (1995). Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM J. Comput.*, 24(2), 247–265.
- Kapoor, S. & Ramesh, H. (2000). An algorithm for enumerating all spanning trees of a directed graph. *Algorithmica*, 27(2), 120–130.
- Kling, D., Elsayed, E. A., & Basily, B. B. (2002). Manufacturing process for folded sheet materials. In *NSF Design, Service and Manufacturing Grantees and Research Conference*, San Juan, Puerto Rico.
- Lang, R. J. (1996). A computational algorithm for origami design. In *Proceedings of the 12th Annual Symposium on Computational Geometry, Philadelphia*, (pp. 98–105).
- Lee, B. & Saitou, K. (2003). Decomposition-based assembly synthesis for in-process dimensional adjustability. *Journal of Mechanical Design*, 125, 464–473.
- Lee, Y. T., Tor, S. B., & Soo, E. L. (1996). Mathematical modeling and simulation of pop-up books. *Computers & Graphics*, 20(1), 21–31.
- Li, B. & Liu, J. (2002). Detail feature recognition and decomposition in solid model. *Computer- Aided Design* 34, 405–414.
- Lin, Y.-L. & Yang, D. C. H. (1994). Automatic development generation for thin-walled objects. *ASME Advances in Design Automation*, 69(1), 367–377.

- Lin, Y.-L. & Yang, D. C. H. (1998). Flat pattern generation of thin-walled objects based on a mechanism theory. *Proceedings of the IMECH E Part B Journal of Engineering Manufacture*, 212(4), 325–334.
- Lipson, H. & Shpitalni, M. (1998). On the topology of sheet metal parts. *Journal of Mechanical Design*, 120, 10–16.
- Liu, H. & Dai, J. S. (2002). Carton manipulation analysis using configuration transformation. In *Proceedings of Institution of Mechanical Engineers*, volume 216, Part C, (pp. 543–555).
- Liu, W. & Tai, K. (2002). Computational geometric modeling and unfolding of 3-d folded structures. In *ASME 2002 Design Engineering Technical Conferences (28th Design Automation Conference)*, Montreal, Canada. paper no. DETC2002/DAC-34046.
- Lu, L. & Akella, S. (1999). Folding cartons with fixtures: a motion planning approach. In *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*, (pp. 1570–1576)., Detroit, MI.
- Lubiw, A. & O'Rourke, J. (1996). When can a polygon fold to a polytope. Technical report 48, Department of Computer Science, Smith College.
- Lyu, N. & Saitou, K. (2003). Decomposition-based assembly synthesis for structural stiffness. *Journal of Mechanical Design*, 125, 452–463.
- Maehara, H. (2000). Can a convex polyhedron have a developable face-cycle? *Theoretical Computer Science* 235, 267–270.
- Mäntylä, M. (1988). *An introduction to solid modeling*. Rockville: Computer Science Press.
- Matsui, T. (1993). An algorithm for finding all the spanning trees in undirected graphs. Technical report, Department of Mathematical Engineering and Information Physics, University of Tokyo, Tokyo.

- Matsui, T. (1997). A flexible algorithm for generating all the spanning trees in undirected graphs. *Algorithmica*, 18(4), 530 – 543.
- Möller, T. (1997). A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2), 25–30.
- Mortenson, M. E. (1997). *Geometric Modeling* (2nd ed.). New York: John Wiley & Sons.
- Mount, D. M. (1992). Intersection detection and separators for simple polygons. In *Proceedings of 8th Annual Computational Geometry*, (pp. 303–311)., Berlin, Germany.
- O'Rourke, J. (1998). Folding and unfolding in computational geometry. In *Proceedings of Japan Conference on Discrete and Computational Geometry*, volume 1763.
- Radin, B., Shpitalni, M., & Hartman, I. (1997). Two-stage algorithm for determination of the bending sequence in sheet metal products. *Journal of Mechanical Design*, 119, 259–266.
- Read, R. C. & Tarjan, R. E. (1975). Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5, 237–252.
- Roth, L. & Wybenga, G. L. (2000). *The packaging designer's book of patterns* (2nd ed.). New York: John Wiley.
- Sahni, S. (1998). *Data Structures, Algorithms, and Applications in C++*. Boston: WCB/McGraw-Hill.
- Schlichenrieder, W. (1997). *Nets of polyhedra*. PhD thesis, Technische University Berlin.
- See Toh, K. H., Loh, H. T., Nee, A. Y. C., & Lee, K. S. (1995). A feature-based flat pattern development system for sheet metal parts. *Journal of Materials Processing Technology*, 48, 89–95.
- Shaffer, C. A. (1997). *A Practical Introduction to Data Structures and Algorithm Analysis*. Upper Saddle River, New Jersey: Prentice Hall.

- Shah, J. J. & Mäntylä, M. (1995). *Parametric and feature-based CAD/CAM*. New York: John Wiley & Sons.
- Shene, C. K. (2002). *CS3621 Introduction to Computing with Geometry Notes*. <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html>, December, 2003.
- Shioura, A., Tamura, A., & Uno, T. (1997). An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3), 678–692.
- Shpitalni, M. & Lipson, H. (2000). 3d conceptual design of sheet metal products by sketching. *Journal of Materials Processing Technology*, 103, 128–134.
- Sun, J. (1999). Folding orthogonal polyhedra. Master's thesis, Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
- Szirmay-Kalos, L. & Márton, G. (1998). Analysis and construction of worst-case optimal ray shooting algorithms. *Computers & Graphics*, 22(2-3), 167–174.
- Tai, K., Liu, W., & Thimm, G. L. (2004a). Unfolding and flat layout design of non-manifold 3d folded structures. In *CAD'04 International CAD Conference and Exhibition*, Pattaya, Thailand. Paper no. 104.
- Tai, K., Liu, W., & Thimm, G. L. (2004b). Unfolding and flat layout design of non-manifold 3d folded structures. *Computer-Aided Design and Applications*, 1(1-4), 439–447.
- Wang, C. H. (1997). *Manufacturability-driven decomposition of sheet metal products*. PhD thesis, Carnegie Mellon University, The Robotics Institute, Pittsburgh.
- Wang, C. H. & Bourne, D. (1996). Compact 2d pattern development from 3d polygonal surfaces. In *ASME Design for Manufacturing Conference (ASME96-DETC)*, Irvine.
- Yetis, F. A. & Saitou, K. (2002). Decomposition-based assembly synthesis based on structural considerations. *Journal of Mechanical Design*, 124, 593–601.

Žalik, B. (2001). Merging a set of polygons. *Computers & Graphics*, 25, 77–88.

Zhu, H. & Menq, C. H. (2002). B-rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition. *Computer-Aided Design* 34, 109–123.