

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9519432

**Automatic labeling, modeling and recognition for line-drawing
interpretation**

Cheng, Tse, Ph.D.

University of Hawaii, 1994

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

AUTOMATIC LABELING, MODELING AND RECOGNITION
FOR LINE-DRAWING INTERPRETATION

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAII IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
IN
ELECTRICAL ENGINEERING
DECEMBER 1994

By
Tse Cheng

Dissertation Committee:
David Y. Y. Yun, Chairperson
Robert Cole
Vassilis Symos
Tong-Jyh Lee
Galen Sasaki

Copyright 1994
by
Tse Cheng
All Rights Reserved

To the memory of my mother, Hui-Qun Xu.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. David Y. Y. Yun, from the bottom of my heart for the years of guidance and encouragement which made this research realizable and enjoyable. I have been particularly impressed by his clear logical thinking and have benefited greatly from his helpful criticisms over every discussion.

I am honored to have Dr. Robert Cole, Dr. Vassilis Syrmos, Dr. Tong-Jyh Lee, and Dr. Galen Sasaki as my dissertation committee members. They have offered me lots of valuable help and useful comments during this work. I also want to express my appreciation to Dr. H. M. Chen Garcia for discussion and suggestions at each research group meeting.

The content of Chapter 3 is primarily based on the results of a project envisioned and supervised by Dr. Yun and supported, in part, by the Department of Business and Economic Development, state of Hawaii through Pacific International Center for High Technology Research (PICHTR). I highly appreciate Mr. Bill Kern for his project management, Mr. Javed Khan, and Mr. Hui Liu for their intellectual contribution to this project. I sincerely thank Mr. Xiangming Feng, Mrs. Grace Ge, Mr. Zong Ling, Mr. Jun Yu and all other members of the Labs. of Intelligent and Parallel Systems (LIPS) and Image/Data Processing Lab. (IDPL) for their valuable assistance and suggestions throughout the whole implementation and dissertation preparation. Special thanks to Charles Lee for his exhaustive program testing and model base construction and Mr. Kui-Yu Chang for helping me with English.

I gratefully acknowledge the 4-year financial support provided by the Industrial Technology Research Institute, Taiwan, Republic of China. Finally, I would like to give my heartfelt thanks to my wife, Orchid Ong. Her love and understanding gave me the strength to complete this work.

ABSTRACT

With the advent of the information era, line-drawing with digital form has become increasingly important in engineering applications. Unfortunately, there is a big media gap between paper and computer. For line-drawings to be useful and manageable they must be archived and understood by computers. However, the mere gathering of those digitized line-drawings certainly does not provide an economic way to store and retrieve. The need has always existed for a real-time system that automates the conversion process to obtain the symbolic descriptions of line-drawing images. Current technology for line-drawing interpretation involves excessive human supervision and is not easily extendible. Furthermore, it also fails to achieve both objectives of minimal model storage requirement and object matching time.

A general paradigm for both 2D and 3D line-drawing interpretation systems is here developed and demonstrated with three major phases: *labeling*, *modeling* and *recognition*. The labeling module extracts a set of features known as symbolic labels of the corresponding objects from the image. These strategically selected labels facilitate automatic modeling and fast recognition dramatically.

For interpreting 2D line-drawings, an automatic symbol segmentation approach for the electrical engineering drawings via the process of image blurring is first devised. A hierarchical neural network is then deployed for symbol modeling and recognition, thereby minimizing human intervention and achieving incremental extendibility capability.

For interpreting 3D line-drawings, a linear-time-complexity polygon-division-based surface extraction algorithm for the projected trihedral objects is proposed. Then, a robust and efficient labeling approach is developed under a Cascaded Constrained Resource Planning (CCRP) model. Its near-linear-time complexity to the classical NP-

complete problems enables extensive usage of symbolic labels for modeling and recognition. Traditional viewer-centered object representation and matching approach requires excessive storage and computation time. Numerous less informative and redundant views are designated to be eliminated and thus gaining the efficiency for model searching. Each valid view is assigned a signature for automatic model base indexing. From the labeled line-drawing, valid-view modeling and multi-view matching are implemented to achieve the goals of lesser storage and faster retrieval time, which combine to realize a real-time and geometrically invariant line-drawing interpretation system.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT.....	vi
LIST OF TABLES	xii
LIST OF FIGURES.....	xiii
CHAPTER 1: INTRODUCTION	1
1.1 MOTIVATION.....	1
1.2 APPROACH OVERVIEWS	7
1.3 ORGANIZATION	9
PART I: TWO-DIMENSIONAL LINE-DRAWING INTERPRETATION ..	10
CHAPTER 2: SYMBOL LABELING AND SEGMENTATION.....	11
2.1 INTRODUCTION	11
2.2 PROBLEM DEFINITION AND RELATED WORK	13
2.3 SYMBOL SEGMENTATION BY IMAGE ABSTRACTION AND BLURRING	18
2.3.1 Image Abstraction.....	20
2.3.2 Image Blurring.....	23
2.3.3 Symbol Segmentation	25
2.3.4 Parallelization of the Overall Segmentation Process.....	26
2.4 EXPERIMENTAL RESULTS	27
2.5 SUMMARY.....	29
CHAPTER 3: SYMBOL MODELING AND RECOGNITION	32
3.1 INTRODUCTION	32
3.2 PROBLEM DEFINITION AND RELATED WORK	34
3.3 FEATURE GENERATION	35

3.3.1 Image Pre-processing.....	36
3.3.2 Invariant Feature Extraction.....	37
3.4 HIERARCHICAL NEURAL CLASSIFIER	38
3.5 EXPERIMENTAL RESULTS	43
3.6 SUMMARY.....	48
PART II: THREE-DIMENSIONAL LINE-DRAWING INTERPRETATION	51
CHAPTER 4: SURFACE EXTRACTION ALGORITHM	52
4.1 INTRODUCTION	52
4.2 PROBLEM DEFINITION AND RELATED WORK	53
4.2.1 Surface Extraction Problem	53
4.2.2 Previous Work.....	54
4.3 POLYGON-DIVISION-BASED SURFACE EXTRACTION	
ALGORITHM.....	55
4.3.1 Algorithms.....	56
4.3.1.1 Find the Outer-most Contour	57
4.3.1.2 Find a Divisible Polygon.....	59
4.3.1.3 Find a Divider	59
4.3.1.4 Do Actual Division.....	60
4.3.2 Complexity Analysis and Comparison	61
4.3.2.1 Jiang and Bunke's Algorithm	61
4.3.2.2 Polygon-division-based Algorithm.....	63
4.3.2.3 Comparison.....	65
4.4 EXPERIMENTAL RESULTS	67
4.5 SUMMARY.....	69
CHAPTER 5: LINE LABELING AND JUNCTION LABELING	70
5.1 INTRODUCTION	70

5.2 PROBLEM DEFINITION AND RELATED WORK	71
5.2.1 Line-Drawing Labeling Problem.....	71
5.2.2 Previous Work.....	72
5.2.3 Motivation and the Proposed Solution	73
5.3 PRELIMINARIES	75
5.3.1 Review of Huffman-Clowes Labeling Scheme	75
5.3.2 Constrained Resource Planning (CRP) Model	77
5.4 LINE AND JUNCTION LABELING BY CASCADED CRP (CCRP) MODEL.....	79
5.4.1 Architecture of CCRP Model.....	79
5.4.2 Prior Knowledge for Selecting Solutions.....	81
5.4.3 Planning-based Labeling Algorithm.....	86
5.4.3.1 Main Function.....	86
5.4.3.2 Pre-classification Rules of Junction Labels.....	88
5.4.4 Data Representation.....	88
5.5 EXPERIMENTAL RESULTS	89
5.5.1 An Example.....	89
5.5.2 Results and Performance Analysis	90
5.6 SUMMARY.....	93
CHAPTER 6: 3D OBJECT MODELING AND RECOGNITION	95
6.1 INTRODUCTION	95
6.2 PROBLEM DEFINITION AND RELATED WORK	98
6.3 PRELIMINARIES	101
6.4 VALID-VIEW OBJECT MODELING BY LABELING	103
6.4.1 Object Modeling Process	104
6.4.1.1 How to Remove Accidental Views	104

6.4.1.2 How to Remove Extreme Views	105
6.4.1.3 Size of Model Base	107
6.4.1.4 How to Remove Redundant Views.....	109
6.4.1.5 An Upper Bound of the Number of Valid Views.....	112
6.4.2 Canonical View List.....	115
6.5 MULTI-VIEW OBJECT MATCHING BY INDEXING	118
6.5.1 Approach Description	118
6.5.2 A New Confidence Computation Model.....	119
6.5.2.1 Rules of Confidence Combination.....	120
6.5.2.2 Advantages of the Model.....	121
6.5.2.3 Influence of Input Sequence	122
6.6 EXPERIMENTAL RESULTS	124
6.7 SUMMARY	126
CHAPTER 7: CONCLUSIONS	128
7.1 CONTRIBUTIONS	129
7.2 DIRECTIONS FOR FUTURE RESEARCH	132
7.3 CONCLUSIONS.....	136
APPENDIX A: 3D OBJECT MODELING AND MATCHING SYSTEM (3DOMMS)137	
APPENDIX B: EXPERIMENTAL MODEL BASE.....	160
BIBLIOGRAPHY	181

LIST OF TABLES

3.1 - Component Error Table.....	48
5.1 - Six Concepts for Both CRPs	81
6.1 - The Accumulated Confidence Table	121
6.2 - The Accumulated Confidence Table for Different Viewing Sequences	122

LIST OF FIGURES

1.1 - Line-drawing Interpretation Process vs. Information Enhancement	2
1.2 - (a) 2D Line-drawing (b) 3D Line-drawings	2
1.3 - Processing Flow of Traditional 2D Line-drawing Interpretation Systems.....	4
1.4 - Processing Flow of Traditional 3D Line-drawing Interpretation Systems.....	4
1.5 - Traditional Viewer-centered Representation of a Tetrahedron.....	5
1.6 - A General Paradigm for Line-drawing Interpretation	6
1.7 - Proposed Approaches for 2D Line-drawing Interpretation	7
1.8 - Proposed Approaches for 3D Line-drawing Interpretation	8
I.1 - A Complete 2D Line-drawing Interpretation System Diagram	10
2.1 - Line Tracing with Symbol Window on a Raw Image.....	14
2.2 - Line Tracing on a Runlength Coded Image	15
2.3 - The Flow of Proposed Symbol Segmentation Approach.....	19
2.4 - The Process Example of Proposed Symbol Segmentation Approach	20
2.5 - Rules for Image Abstraction (a) 4-pixels (b) 3-pixels (c) 2-pixels (d) Boundary Conditions for (c)	21
2.6 - Image Abstraction (a) 256*256 original (b) 128*128 (c) 64*64 (d) 32*32 (line width = 8)	22
2.7 - Image Blurring (a) $f = 0.5s$ (b) $f = 0.8s$ (c) $f = s$ (where f is the defocus distance and s is symbol size)	24
2.8 - Symbol Segmentation Around the Local Maximum after Blurring.....	26
2.9 - Image Abstraction on (N+1) Processors for M*M Line-drawing Image	27
2.10 - Experimental Results on Symbol Segmentation.....	28
2.11 - Some Test Results of the Text and Picture Segmentation.....	30

2.12 - A General Image Segmentor.....	31
3.1 - Image Pre-processing	36
3.2 - A Two-layer Perceptron Network	39
3.3 - Hierarchical Neural Network Classifier	42
3.4 - SRS Basic Architecture	43
3.5 - The Detection Rates for Both Training and Test Sets.....	44
3.6 - Hidden Layer Width Selection	45
3.7 - Symbols Used in SRS system (34 symbols, 23 classes and 4 groups).....	47
II.1 - A Complete 3D Line-drawing Interpretation System Diagram.....	51
4.1 - The Surface Extraction Problem	54
4.2 - A Step-by-step Example of Polygon-division-based Algorithm.....	56
4.3 - Three Different Orders of Edge $\langle SJ, J_2 \rangle$	58
4.4 - Two Extreme Cases (a) Size of CEL \approx #E (b) Size of CEL \approx 0.....	65
4.5 - An Example.....	65
4.6 - Find Outer Contour	66
4.7 - Find_Divider(\cdot) and Do_Divide(\cdot).....	67
4.8 - Performance of the Surface Extraction Algorithm.....	68
5.1 - Different Junction Interpretation Results Different Line Labels	74
5.2 - Edge Definitions of T- and W-junctions.....	75
5.3 - Huffman-Clowes' Labeling Dictionary	76
5.4 - The Corresponding Junctions in Realistic Line-drawings.....	77
5.5 - Configuration of the CRP Model	78
5.6 - Architecture of Cascaded CRP Model	80
5.7 - Case 1: One Y- and Three W-junctions.....	83
5.8 - Case 2: One T- and Three W-junctions.....	83
5.9 - Case 3: Four W-junctions	84

5.10 - Extreme View of a Perspective Projection	84
5.11 - Contradictory Examples	85
5.12 - A Y-L Pair for a Convex or Concave Y-junction	85
5.13 - Y-L Pair for a Y-junction with Different Labeling.....	86
5.14 - An Contradictory Example for Any Trihedrons.....	86
5.15 - Flowchart of Cascaded Junction/Line Labeling CRP	87
5.16 - Thresholds for Junction Type Pre-Classification.....	88
5.17 - The Data Structure Used for Junction/Line Labeling Cascaded CRP.....	89
5.18 - Example of (a) Original Drawing (b) Drawing After Pre-processing.....	90
5.19 - Labeled Line-drawing with No Backtrack.....	91
5.20 - Labeled Line-drawing with One Backtrack	91
5.21 - Performance of Labeling Algorithm for 12 Modeled Line-drawings	92
5.22 - Performance of Labeling Algorithm for Different Problem Sizes.....	93
6.1 - Two Similar Objects	97
6.2 - 3D Viewing Space with an Example of L-shaped Object.....	99
6.3 - Some Extreme Views of the Object in Figure 6.1(a)	102
6.4 - (a) Labelable View (b)-(c) Accidental (Non-labelable) Views.....	102
6.5 - An Imperfect Line-drawing.....	105
6.6 - A Y-junction Has At Least One Neighboring W-Junction	106
6.7 - (a) $f = 6, v = 8, e = 12$ (Cube); (b) $f = 4, v = 4, e = 6$ (Tetrahedron)	109
6.8 - (a) $f = 8, v = 12, e = 18$ (L-shaped); (b) $f = 9, v = 14, e = 21$	109
6.9 - Valid-View Object Modeling Process	111
6.10 - Right-angle Trihedrons	112
6.11 - Two Topologically Different Views but Have Same Signature.....	116
6.12 - Example of Model Base.....	120
6.13 - Valid Views of L-shaped Object	125

6.14 - Performance of the Simulated 3D Object Recognition System.....	126
7.1 - A Multi-object Scene.....	133
7.2 - Three Different Interpretations of a Line-drawing.....	134
7.3 - Extended Junction Types for General Polyhedral and Curved Objects.....	135
A.1 - Screen Layout	138
A.2 - Reading obj7.2	143
A.3 - Inputing 3-dimensional T.....	146
A.4 - Browse.....	148
A.5 - Browsing T	150
A.6 - Extracting Faces of T.....	151
A.7 - Labeling T.....	152
A.8 - obj9.2.....	154
A.9 - Canonical View for object 9.....	155
B.1 - Experimental Model Base	161

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Line-drawing remains one of the essential representations for data processing, transmission, and archiving. Drawings described in an electronic format are known to facilitate storage, retrieval and management as well as easy manipulation and reproduction. However, a vast amount of existing drawings that are not yet stored in this format need to be computerized. Since CCD (Charge-Coupled Device) cameras and scanners have become immensely popular and inexpensive, drawings can be captured into a computer as bitmaps. Consequently, the simplest way to store and retrieve the line-drawings is to create a large image database. However, the mere gathering of image data certainly does not provide an intelligent and fast retrieval system to a large and complex collections of images. In order for those line-drawing images to be useful, they must be "understood" or "described symbolically" by computer. However, current solutions involve too much human re-entry or digitizing effort. Moreover, not only special knowledge of the drawings and computer software are required in order to perform such task, it is also time-consuming and labor-intensive. It is envisaging a need for developing an *automatic* and *intelligent* approach that uses *minimal storage* for *fast information retrieval* to a vast usage of those line-drawings in the near future. Figure 1.1 shows the four line-drawing interpretation processes and the corresponding input and output information it involves at each stage. This research primarily aims at developing some faster labeling and recognition techniques for interpreting both 2D and 3D line-drawings.

Since the recognition domain is pre-determined, the objects to be recognized are modeled beforehand.

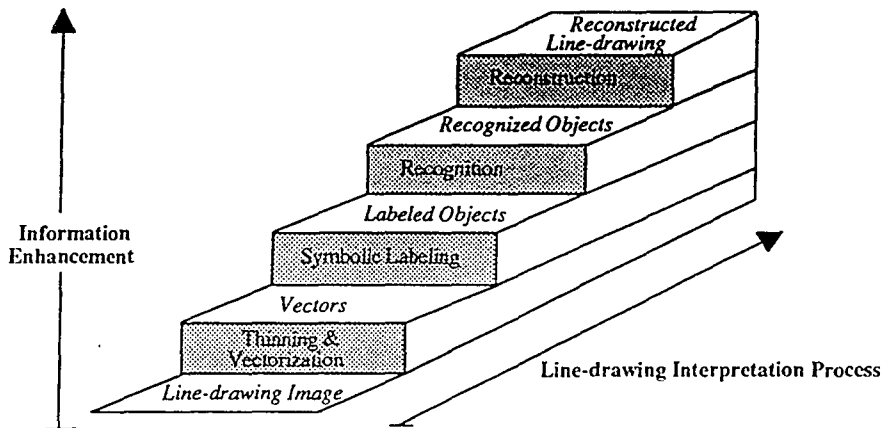


Figure 1.1 - Line-drawing Interpretation Process vs. Information Enhancement

A given line-drawing contains a wealth of information which can however be categorized into two main types: 2D and 3D. Examples of popular 2D line-drawings are engineering designs such as electrical circuit diagrams, mechanical parts drawings and maps. Symbols are usually the most important information contained in these drawings. Therefore, symbol recognition will be the main focus for the overall automated 2D line-drawing interpretation system. On the other hand, as 3D objects are usually recognized from their 2D projections (images), a typical 3D line-drawing is thus just an edge mapping of the projections of 3D objects onto the 2D viewing plane.

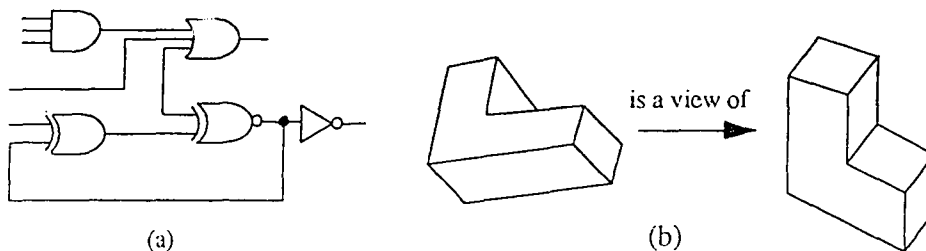


Figure 1.2 - (a) 2D Line-drawing (b) 3D Line-drawings

For example, as depicted in Figure 1.2, engineering drawings (e.g. circuit diagrams, architecture layouts, etc.) as well as maps, are 2D line-drawings; and information embedded in the silhouettes of 3D object scenes are 3D. Due to the variety of information contained in 2D and 3D line-drawings, there are different ways to label, model, and recognize the objects in the drawings. The drawing on the left in Figure 1.2 is a 2D electrical logic diagram containing one each of the following: AND, OR, XOR, XNOR, and INV (inverter) gates. The drawing on the left of Figure 1.2(b) has 4 visible surfaces, 13 line segments, and 10 junctions. In addition, it is a view of the 3D object shown at drawing on the right.

Traditionally, the labeling or segmentation of a 2D line-drawing has mostly been based on a thinned image or vectors. Most of the previous solutions look for objects (symbols) by tracing the connecting lines on the vectorized line-drawing and analyzing or segmenting the regions whenever some special (pre-defined) features have been encountered. In the mean time, the recognition is usually accomplished by graph-based structure pattern analysis, which includes decision tree and graph matching techniques. Figure 1.3 shows the processing flow of traditional 2D line-drawing interpretation systems. Those traditional methods are usually inefficient and hard to parallelize, as well as hard to accommodate new objects or applications. In this research, segmentation and recognition processes are performed on raw image. Symbols are segmented by blurring mechanism, which looks for each aggregation of pixels on the abstracted and blurred image. Moment features of each symbol image are extracted and input to a hierarchical neural network for a fast and adaptive classification.

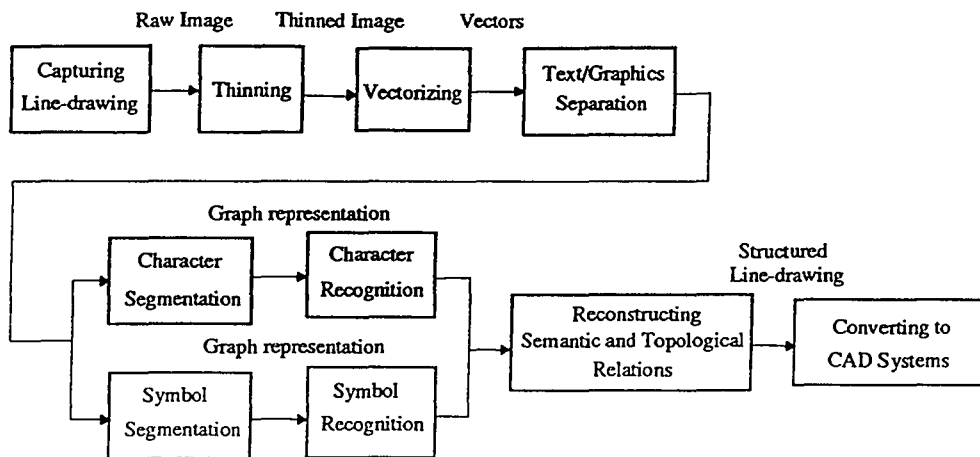


Figure 1.3 - Processing Flow of Traditional 2D Line-drawing Interpretation Systems

Most of the 3D object recognition systems are model-based, where features extracted from 2D line-drawings are matched with stored 3D object models. A 2D line-drawing of a 3D object scene is obtained from the intensity image after edge detector and line tracer. Since many satisfactory approaches have been proposed to produce near-perfect line-drawings [GuHu87] [LieC90] [Huan93], one may assume that all of inputs are perfect line-drawings. A perfect line-drawing is a line structure without dangling points or line segments and no missing corners or line segments.

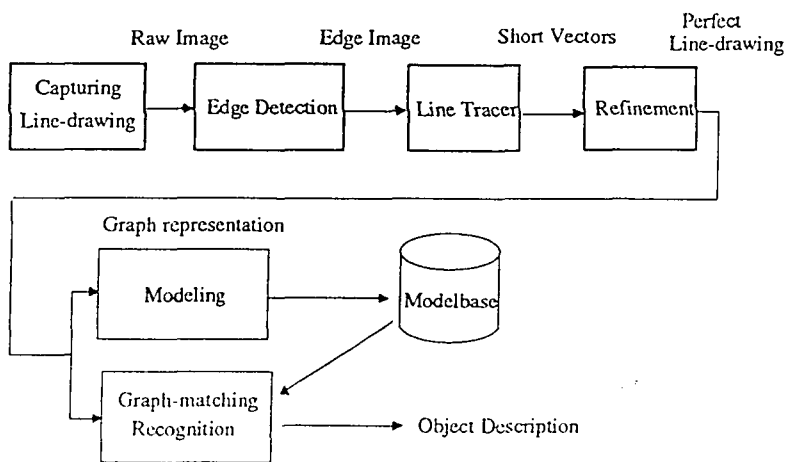


Figure 1.4 - Processing Flow of Traditional 3D Line-drawing Interpretation Systems

A traditional 3D object recognition system based on its 2D line-drawings is shown in Figure 1.4. In this research, it is assumed that the input is a perfect line-drawing and the object domain is limited to the trihedral. A trihedral object is a polyhedron which has exactly three plane surfaces meeting at each vertex.

Presently, *viewer-centered*, or *multi-view object representation*, is one of the most popular 3D object modeling approaches. In this approach, a 3D object is represented by many (often a large number, e.g. 71 for an L-shaped block) 2D projections (*views*). Thus, the object in any particular view can, then, be matched directly with the views stored in the model base. However, this traditional method of model storage and matching frequently suffers from the problems of excessive storage and computational requirements. Given a view of an unknown object, the cost of searching, even for a simple object, in a large model base is very high. Figure 1.5 shows a traditional viewer-centered representation of a tetrahedron [SteB90]. Fourteen views are connected as a graph. Each arc represents direct path from one view to another view (an visual event), where a topological difference appears. To show these differences, four surfaces are colored with different shadings.

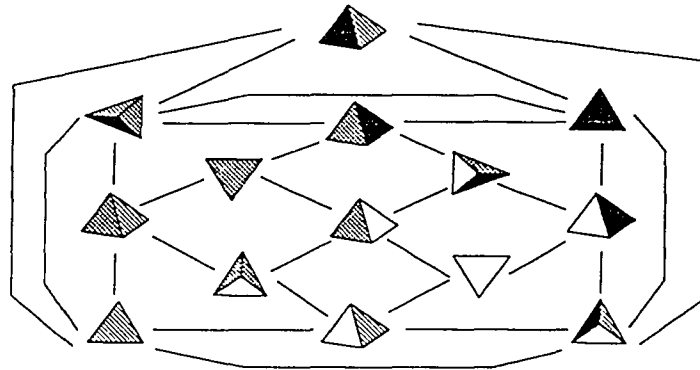


Figure 1.5 - Traditional Viewer-centered Representation of a Tetrahedron

In addition, since the line-drawing of 3D projection is a planar graph, the traditional matching is usually accomplished by a time-consuming graph-matching

approach. In this research, a set of symbolic features, which includes the visible surfaces, the type of junctions and the convexity/concavity of edges, of line-drawing is efficiently extracted. These linear-time-performance feature extraction algorithms and symbolic representation enable to construct a rapid and reliable 3D recognition system via indexing multiple input views.

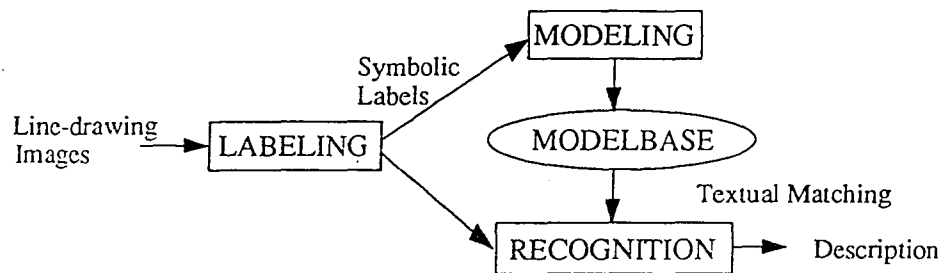


Figure 1.6 - A General Paradigm for Line-drawing Interpretation

Based on an extensive literature search, it is concluded that a general paradigm needs to be established for both 2D and 3D line-drawing interpretation, shown in Figure 1.6. Current solutions for interpreting either 2D or 3D line-drawings involve much domain knowledge in model encoding and recognition of the input image. This, in turn, is done by an inefficient graph-matching approach that usually takes an exponential time complexity for each pair of input and model graph. The above reasons motivated us to propose and develop a set of general but highly efficient computational approaches that will overcome all of the shortcomings. The labeling process involves feature-extraction and grouping. The output of the labeling module is a set of features also known as symbolic labels of the corresponding symbols or objects. The modeling process automatically encodes these object features into the model base. The input to the recognition module is also a set of symbolic labels, however, the module performs recognition by matching the input labels to the encoded object labels in the model base.

1.2 APPROACH OVERVIEWS

Three major components for 2D and 3D line-drawing image interpretation systems have been first developed and demonstrated in this dissertation. As shown in Figure 1.5, they are: *labeling*, *modeling* and *recognition*. For the 2D case, this dissertation first formulates an automatic object (symbol) segmentation (labeling) approach for the electrical engineering line-drawing image by image abstraction and blurring. Next, a set of translation-, scaling- and rotation-invariant features are designated for symbol modeling and recognition. A hierarchical neural network approach for symbol modeling, and recognition is then proposed to achieve the goals of incremental extension and minimal human involvement and storage. The above 2D line-drawing labeling, modeling and recognition approaches are validated by the experiments of circuit diagrams (schematics). Figure 1.6 shows a mapping of our proposed 2D line-drawing labeling, modeling and recognition approaches to the general paradigm.

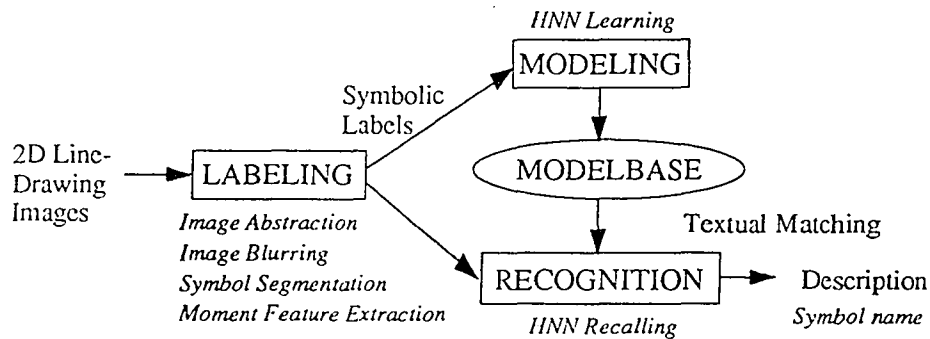


Figure 1.7 - Proposed Approaches for 2D Line-drawing Interpretation

In the second part of this dissertation, a linear-time complexity surface extraction algorithm is proposed. Unlike other previous approaches, this algorithm divides the object's exterior contour into a sets of minimal regions, which correspond to visible surfaces of the 3D objects in the scene. Recursive divisions will be carried on if the

divided polygons are still divisible. A robust and efficient 3D line-drawing labeling approach has been proposed and tested. Traditional line labeling approach only labels lines under the assumption that junction types were known in advance, whereas the proposed approach labels lines and junctions alternatively and is developed under a Cascaded Constrained Resource Planning (CRP) model. A near-linear-time computational complexity of this approach has been verified by a number of test line-drawings from the experiments.

Based on the labeled line-drawing images, a novel valid-view 3D object modeling and multi-view constant-time complexity matching approach has been developed, which is able to achieve less storage and faster retrieval time. Instead of using the time-consuming (usually an NP-complete) graph-matching approach, in this research, a set of invariant features are extracted and the signature of each valid view is generated for automatic object matching by symbolic indexing. A 3D Object Modeling and Matching System (3DOMMS) for the recognition of trihedrons has been implemented which not only demonstrates the feasibility of the proposed approaches but also provides an interactive modeling environment to the users. Figure 1.8 shows a mapping of our proposed 3D labeling, modeling and recognition approaches to the general line-drawing interpretation paradigm.

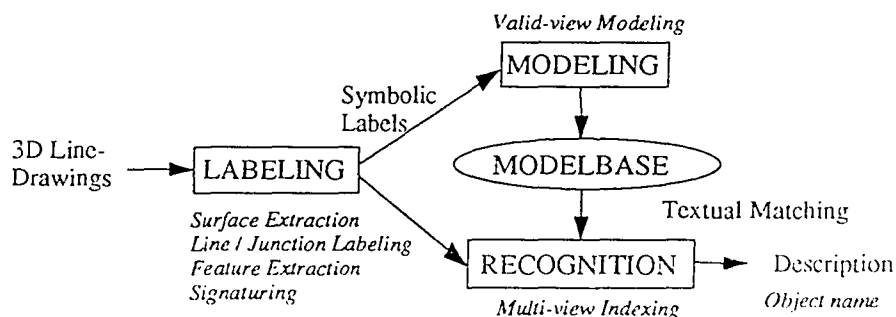


Figure 1.8 - Proposed Approaches for 3D Line-drawing Interpretation

1.3 ORGANIZATION

This dissertation is composed of two parts. Part I, Chapter 2 and 3, contains all three major components which are developed by the new approaches for understanding 2D line-drawing images. Chapter 2 describes the symbol segmentation approach by image abstraction and blurring for electrical engineering drawings. Chapter 3 presents a hierarchical neural network approach for a successful recognition of 34 different types of logic gates. In each chapter, the problem is defined and related work are summarized before introducing the proposed approach. Experimental results and analysis are given at the end of each chapter. In Part II, Chapter 4, 5, and 6, a set of new labeling, modeling, and recognition approaches for the interpretation of 3D line-drawings are given. The fourth chapter will present a new surface extraction algorithm, which can extract surfaces from a given projection of any trihedron in linear time. In chapter 5, a robust and efficient line-drawing labeling algorithm, which labels lines and junctions alternatively, is presented. Its effectiveness, in terms of average execution time, seems to enable extensive usage of the labeling process for 3D object modeling and recognition. In chapter 6, a labeling-based valid-view object modeling and multi-view matching (recognition) approach will be shown. In that chapter, some proofs and experimental results will demonstrate the advantages of these new approaches over other traditional object modeling and matching techniques in terms of storage usage and computational efficiency. Conclusions are drawn in the last chapter, which details the contributions of this research and provides suggestions of future enhancements. A simulated 3D object modeling and matching system (3DOMMS) is described detailedly in Appendix A. The experimental model base which includes several typical 3D objects is given in Appendix B. Statistical information such as the number of valid views and the number of backrackings of individual object are also listed. Finally, a bibliography follows at the end of this dissertation.

PART I

2D LINE-DRAWING INTERPRETATION

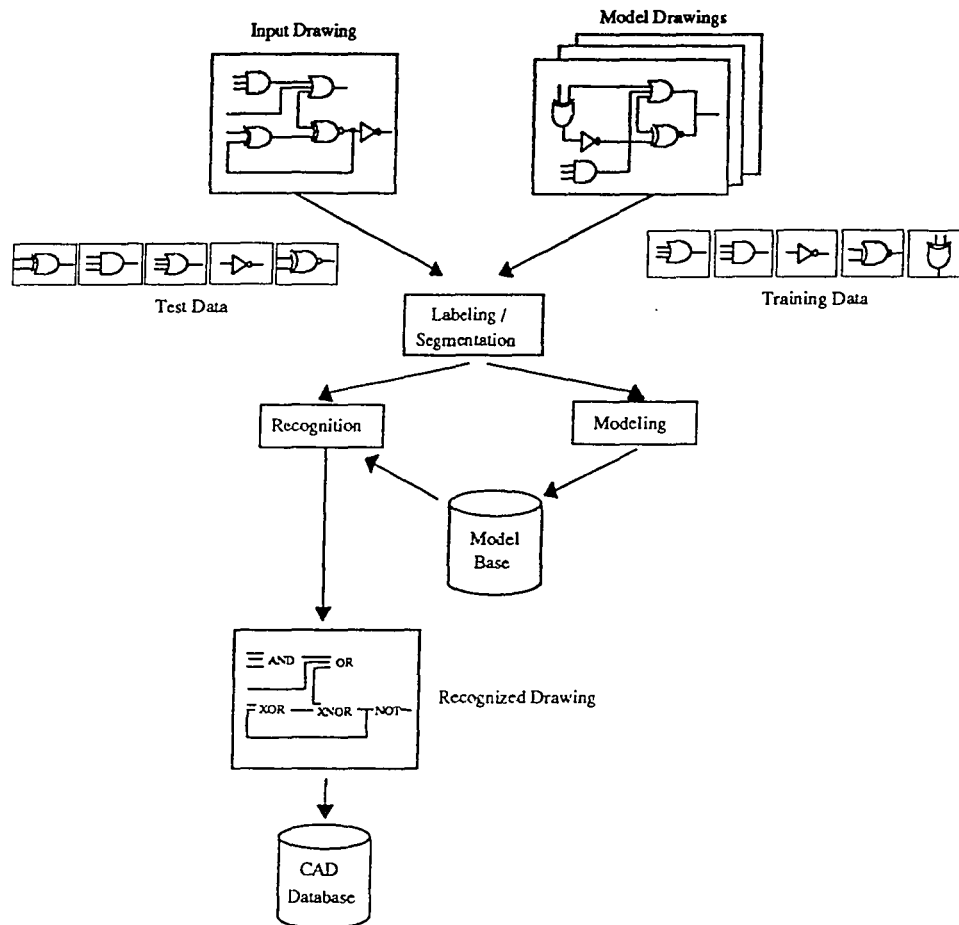


Figure I.1 - A Complete 2D Line-drawing Interpretation System Diagram

Companies using CAD (Computer Aided Design) systems often face the problem of converting existing or hand-drawn designs into digital format. Conventional human-reentry or digitizing work is often time-consuming and usually error-prone. A new industrial need is in performing the recognition task leading from the line-drawings to its description automatically and thus enhancing the capabilities of existing CAD systems.

CHAPTER 2

SYMBOL LABELING AND SEGMENTATION

2.1 INTRODUCTION

Drawings stored in electronic formats are easier to manage, modify and reproduce. However, a vast amount of existing drawings created by traditional means remain to be computerized. For a long time, one of the industrial needs has been an automatic engineering drawing conversion system capable of replacing conventional human-reentry or digitizing work that is labor-intensive, time-consuming and usually error-prone.

The widespread introduction of scanning devices to industries has established the first step of conversion paper drawings into bitmaps which can be stored by computer media. However, the mere gathering of those bitmaps certainly does not provide an economic way to store and retrieve. Second level of conversion involves extracting line structures from line-drawing images dominates the Automatic Conversion of Image Document (ACID) market presently. Many thinning and vectorization algorithms have been proposed [CFMP84][Pav186] to convert the bitmaps into line structures for lesser storage and easier edition. However, the output of these systems still does not reach the level of interpretation required for (Computer Aided Design) CAD input. A need for higher level of interpretation such as symbol and text recognitions is then emerged obviously.

Recognizing the symbols of an engineering drawing in bitmap format is a critical step towards computer interpretation of line-drawings. For engineering drawings not generated by a computer or where the original computer file is accidentally destroyed,

they can be scanned and stored as bitmaps. To understand the bitmap or to restore the original data format, the symbols in the drawings must be segmented first.

Segmentation is a process of extracting, from a given image, a set of pixels that have pre-specified meanings or associated utilities. Symbol segmentation is one of the most crucial tasks during the image conversion process. A good segmentation will result in a good recognition as well as a higher conversion accuracy and efficiency. Symbols are different from texts or characters in most of the engineering drawings because they are usually inter-connected by either straight or cursive lines. In engineering or science applications, a variety of graphical symbols are used in diagrams or drawings to express human ideas or intentions which are otherwise difficult to describe in plain words. Previous works involve thinning, line-tracing, and structural analysis, which are the three most common steps for segmenting symbols from line drawings. Approaches based on the above methodology usually suffer from poor performance on noisy images, slow sequential operation (line tracing step) and little expandability to accommodate new applications.

The rest of this chapter is organized as follows. Problem definition and related work is given in section 2.2. A three-stage symbol segmentation technique is introduced in section 2.3. The first step, called image abstraction, reduces the size of the working bitmap while maintaining the geometry of each symbol. The second step utilizes image blurring to locate the possible positions of each symbol. Finally, each symbol is segmented around the detected positions. This approach works directly on the raw images as neither thinning nor vectorizing is needed. Furthermore, the parallelization of the overall process is also proposed. Experimental results are shown in section 2.4. Finally, a discussion and possible extension of this approach are given in the summary section.

2.2 PROBLEM DEFINITION AND RELATED WORK

A symbol is defined as a collection of line segments or graphical primitives representing a specific meaning or possessing a particular property with respect to its position in the drawing. From the geometrical shape of a symbol, individual characteristics can be clearly recognized. For example, a rectangle in Electrical Engineering Line-Drawings (EELDs) represents a flip-flop. Furthermore, from its placement, one may understand the topological relationship to its neighbors, ultimately understanding the content of the overall drawing. For example, if the rectangle has one terminal on the top, it is a D-type flip-flop. Symbols used in EELDs are divided into two major categories: close-loop symbols and open-loop symbols. AND, OR, XOR, ...etc. are called close-loop symbols because all symbols of this category have drawings that are connected and formed a close loop. On the other hand, BAT (battery), CAP (capacitor), GND (ground), ...etc. are open-loop symbols since their drawings are not fully connected.

In general, symbol segmentation for EELDs can be divided into two major categories: *line-tracing with symbol window* and *line-filtering*. The first approach usually starts from an endpoint of a line segment and traces the line in the scope of a symbol window. The tracing stops after it has detected a possible symbol or reached another endpoint. An analysis then proceeds to find the exact symbol location and size. Segmentation follows after the analysis shows a high probability that the symbol is completely inside the window.

The second approach, *line-filtering*, uses a totally different idea to segment symbols from EELDs. *Line-tracing* looks for a symbol, but *line-filtering* searches for connecting lines and removes them from the drawings. All previous solutions are based on one assumption: the length of the line connecting two symbols is much greater than the

lengths of the line segments within each symbol. A pre-defined threshold removes the longer straight lines after vectorization or line classification. However, this assumption is very dangerous and impractical for real cases. The reasons will be explained in the following subsections.

Youji Fukada proposed a line tracing approach, Figure 2.1, which tracks connecting lines that have *thickness* using a line sensor [Fuka84]. The line sensor is made up of three parts, namely a front sensor and two side sensors. Once either or both side sensors detect picture pixels, this area (significant area) will be examined in detail because it may contain a symbol. Otherwise, cracks, slants or Y-type junctions are checked to determine whether the line being traced is a connecting line or a part of a symbol. For given drawings, eight constraints are used for this algorithm of which some, such as line width and symbol size, are either unnecessary or too restrictive to the designers.

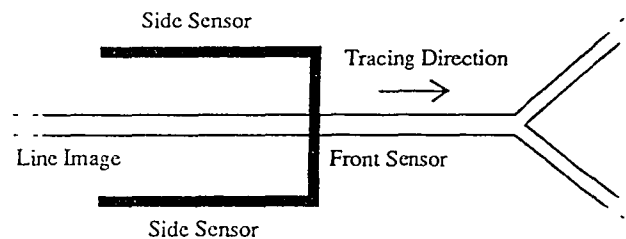


Figure 2.1 - Line Tracing with Symbol Window on a Raw Image

Furuta et. al. proposed a method for symbol segmentation by tracking the line on *runlength coded image* [FuKE84], shown in Figure 2.2. A symbol is deemed to exist and line tracking is suspended after one of the following four conditions is identified during line tracking: 1) length of run > maximum line width; 2) length of run > 1.5 * average run length of line being tracked; 3) existence of bending points; 4) non-existence of a run that

has common run-direction domain with a run of a previous row. A symbol frame is formed after a continuing line is detected. The following two conditions are used to check the existence of a continuing line: 1) length of run $\geq 1.5 * \text{average run length of line being tracked}$; 2) row of runs $\geq \text{minimum number of runs to be defined as a continuing line}$. A symbol is segmented after the line tracer has detected a continuing line.

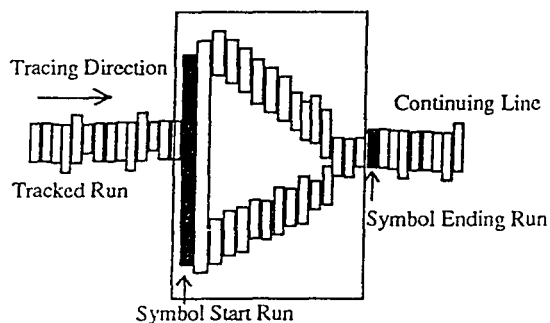


Figure 2.2 - Line Tracing on a Runlength Coded Image

The third type of line tracing methods is based on a *thinned image*. This is the most popular image representation for structural analysis of line patterns. Groen et. al. used global analysis to divide the thinned line drawing image into objects (symbols) and interconnections [GrSS85]. The candidate objects are those skeleton parts enclosing a connected background component. They are found by component labeling of the background, expansion of the background component and taking the AND function of the expanded background component and the skeleton. Lee applied Groen's top-down global analysis approach to segment different components in EELDs [Lee92]. A similar approach to detect symbols on thinned image was proposed by Okazaki et. al. [OKMT88]. Loop-structure-based (close-loop) symbol is first extracted and a minimum region for analysis (MRA) is then formed for symbol identification. In the second phase - loop-free symbol segmentation, symbols are segmented and recognized by first searching

key feature points such as an end point for a GROUND symbol, a corner point for a RESISTOR or a functional rectangle for flip-flops. These structural features are relatively easy to be detected and extracted by performing line-tracing from those key points.

A rule-based segmentation approach was proposed for understanding chemical plant engineering drawings by Harada [Har85]. In this approach, the image is first *vectorized*, and a hierarchical decision tree is followed to classify vectors into characters, lines, and symbols according to 40 segmentation rules. Loop information, connection information, and the density of short vectors are used for classification. Hamada proposed a hypothesis building and validation method to select and verify the symbol area where constituting a loop structure or neighboring to a text component [Hama93]. This method also works on the vectorized line-drawing.

In contrast to the above line tracing approaches for symbol segmentation, line filtering approaches remove (interconnecting) line structures from the drawing instead of finding symbols along the lines. Clement [Clem81] proposed a method to remove lines in a *multi-resolution* framework. The line structure will be removed and symbols will become isolated blocks by averaging neighboring pixels from fine to coarse resolution. The merging window size and cut-off value are two experimental parameters to be optimized. In his paper, a five-by-five window gives a reasonably good result and twelve seems to be an optimal threshold value. Meaning that a pixel in a coarse level is '1' while the corresponding window in the fine level has more than twelve black pixels. British engineering drawing standard was used in his work which constraints that two parallel lines should not be closer than 1 mm. Line width and symbol size are also restricted and depended on the merging window size and cut-off value.

Bley's method [Bley84] splits and merges runlength codes for lengthy straight-lines extraction. After *runlength code* generation, a picture graph is built for split-and-merge

iterations. The line elements are computed by two different methods. First, primary component graphs are split or merged into classified line elements which describe the dominant large lines of the drawing. The line elements are then connected to each other, forming "connected components" larger than letters. Therefore, it is easy to classify line elements and objects (or letters). Second, the details are analyzed within the context of dominant lines using production systems.

Lin [LSMS85] has proposed a grid merging technique for logic symbol segmentation. In his paper, he first divides the image into small rectangular regions and then extracts long straight lines by merging neighboring grids. At first, the image is divided by a regular grid. Every square plane of the grid is called an unit mesh. An unit mesh contains plural pixels (e.g., several tens to several hundreds). Combining all of the possible appearances of the five regular patterns on the four sides of a unit mesh border, 47 characteristic patterns are derived. Horizontal and vertical long straight line segments are then extracted by analyzing the types and connectivity of neighboring line patterns.

Connecting lines are treated as noise while segmenting symbols. Therefore, the line-filtering method is a promising approach that removes noise as much as possible before recognition. Nevertheless, it is quite difficult to distinguish open-loop symbols from connecting lines. Most of the current solutions are based on a fact that connecting lines are long straight lines and symbols are composed of short straight lines. As a consequence, those previous approaches do not work well if the drawings contain large symbols and short connecting lines.

In summary, previous symbol segmentation approaches identify symbols from localized or narrowly focused manipulations of pixels. They are sequential in nature and do not perform well for open-loop symbols. In addition, they are usually sensitive to the

symbol size, average length of connecting lines and noise (such as spurs) caused by the thinning process.

2.3 SYMBOL SEGMENTATION BY IMAGE ABSTRACTION AND BLURRING

Instead of using localized pixel-tracing to search for spatial relationships, segmentation is performed by global extraction of coarse associations among groups of pixels. *Image blurring*, which first proposed by F. Bergholm for edge focusing [Berg87], is a mechanism of blending influence of neighboring pixels into a central pixel. The influence of every other pixels to a particular concerned pixel is in the reverse proportion of their distances. Any bounded symbol is usually an aggregation of its component pixels. Through blurring, such an aggregation of nearby pixels will generate a local intensity maximum around the geographical center of the symbol and other unrelated pixels will fade away due to either the remoteness of the distance or the lack of enough black pixels [Chen92].

The method proposed for segmenting electrical engineering symbols is comprised of three stages. First, an image abstraction process is used to reduce the size of the raw image, thereby reducing the computation time in subsequent steps, while maintaining the characteristic shape of each symbol. The abstracted image is then blurred with an appropriate distance of defocusing in the second stage so that a local peak of intensity will appear within the scope of each symbol. In the third stage, a window with the appropriate size for a particular kind of symbol is superimposed on the given drawing image and moved about around each local peak in an attempt to cover the entire symbol by maximizing the number of black pixels lying within the window. After three steps, all symbols with size of smaller or equal to the moving window can be properly segmented within the original image. Compared to the traditional segmentation approaches, e.g. line

tracing that concentrates on the details of each pixel, this three-stage method focuses directly on global information of the image, ignoring a large amount of irrelevant details so that neither thinning nor vectorizing is needed (Figure 2.3).

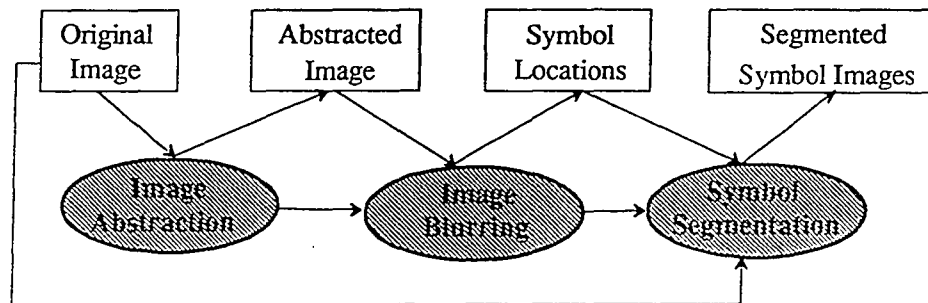


Figure 2.3 - The Flow Diagram of Three-stage Symbol Segmentation Approach

In addition, this algorithm can also be efficiently implemented on a multi-processor computer. Results from segmenting logic-gate symbols in electrical schematics demonstrated in this chapter will show that this method can detect and segment not only close-loop symbols (e.g. AND gates) but also open-loop symbols (e.g. capacitors and resistors). Therefore, the feasibility and practicality of using image abstraction and blurring techniques to segment symbols of an engineering drawing image have been established. Figure 2.4 shows an example of results obtained after each main steps of the proposed method. Figure 2.4(a) is an original raw image containing 3 symbols (one OR and two AND gates). Figure 2.4(b) is the abstracted image, which is small-sized and thinned image of the original one. Figure 2.4(c) shows three symbol locations detected by image blurring technique and Figure 2.4(d) is the results of 3 segmented symbols.

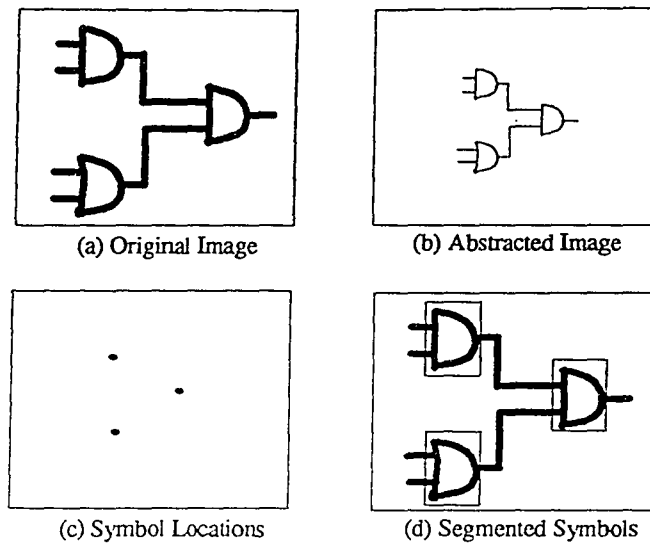


Figure 2.4 - An Example of the Different Stages

2.3.1 Image Abstraction

Before doing image blurring on the large matrix of raw bitmap images, an abstraction process is performed. This abstraction will greatly reduce the size of the raw image while still retaining the original shape of each symbol in an abstracted image. The whole abstraction is processed in a series of steps, in the process of generating a pyramid (hierarchical layers) of images. The size of the image is reduced to a quarter after each step and the number of abstractions depends on the prior knowledge of the image, particularly the average line width.

Assuming that the average line width in the raw image is d . After one abstraction step, the average line width in the new image is reduced by half. In order to keep the original geometry of the symbols, one basic requirement is that any chosen line in the raw image should have a corresponding line in the final abstracted image. Since the minimum line width in a bitmap format image is one, the number of abstraction steps should not exceed $\log_2 d$.

	Before Abstraction		After Abstraction																	
(a)	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td></tr></table>	1	1	1	1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1												
1	1																			
1	1																			
1																				
(b)	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td>1</td><td></td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td></td><td>1</td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>1</td></tr><tr><td>1</td><td>1</td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td></td></tr><tr><td>1</td><td>1</td></tr></table>	1	1	1		1	1		1		1	1	1	1		1	1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1
1	1																			
1																				
1	1																			
	1																			
	1																			
1	1																			
1																				
1	1																			
1																				
(c)	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td></td></tr><tr><td></td><td>1</td></tr></table> <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>1</td></tr><tr><td>1</td><td></td></tr></table>	1			1		1	1			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1								
1																				
	1																			
	1																			
1																				
1																				
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td></td><td></td></tr></table> if its top neighbor is	1	1			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td></td></tr><tr><td>1</td><td>1</td></tr></table> then			1	1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1								
1	1																			
1	1																			
1																				
		else	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>																	
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>1</td></tr><tr><td></td><td>1</td></tr></table> if its right neighbor is		1		1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td></td></tr><tr><td>1</td><td></td></tr></table> then	1		1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1								
	1																			
	1																			
1																				
1																				
1																				
		else	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>																	
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td></td></tr><tr><td>1</td><td>1</td></tr></table> if its bottom neighbor is			1	1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td></td><td></td></tr></table> then	1	1			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1								
1	1																			
1	1																			
1																				
		else	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>																	
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td></td></tr><tr><td>1</td><td></td></tr></table> if its left neighbor is	1		1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>1</td></tr><tr><td></td><td>1</td></tr></table> then		1		1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1								
1																				
1																				
	1																			
	1																			
1																				
		else	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>																	
(d)	if <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td></tr><tr><td></td><td></td></tr></table> appears at top image boundary then	1	1				<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>													
1	1																			
	if <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>1</td></tr><tr><td></td><td>1</td></tr></table> apperars at right image boundary then		1		1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>													
	1																			
	1																			
	if <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td></td></tr><tr><td>1</td><td>1</td></tr></table> appears at bottom image boundary then			1	1		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>													
1	1																			
	if <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td></td></tr><tr><td>1</td><td></td></tr></table> appears at left image boundary then	1		1			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td></tr></table>													
1																				
1																				

Figure 2.5 - Rules for Image Abstraction (a) 4-Pixels (b) 3-Pixels (c) 2-Pixels (d) Boundary Conditions for (c)

In each step of abstraction, every four neighboring pixels forming a square will be simplified into one pixel, thus reducing the sizes in each dimension by half. A set of rules is established to determine whether the abstracted pixel is black or white. For example, among 4 raw pixels, if there are more than 2 black pixels, the resulting pixel will be black; if there are more than 2 white pixels, the resulting pixel will be white. More detailed rules are used if there are equal number of black pixels and white pixels. If two pixels of the same color lie on either diagonals, the resulting pixel will be black; otherwise, the two black pixels concluded to be adjacent and pixels in the neighboring square will be

examined if the two pixels are not at the image boundary (Figure 2.5). If they are both black, the resulting pixel will be black; If not, the resulting pixel will be white. For two consecutive pixels at the image boundary, the abstracted pixel is set to white if they are isolated from the symbol image. For example, if two black pixels appear at the top of the boundary image and no pixels appear below them, they are certainly not connected to the original image and will be treated as noise.

Unlike thinning methods which take up a lot of computation time to maintain the precise shape of the original image, this method is simple and beautiful because the exact shape of the image need not be maintained. As long as pixels from a symbol is still close to each other in an abstracted image, they will meet the requirement of the subsequent blurring step. Obviously, the above simple heuristic is good enough to fulfill the requirements elegantly. An example of an abstracted image is shown in Figure 2.6. As the raw image has an average line width of 8, three abstraction steps were carried out. The abstracted image clearly retains the geometry of the original image.

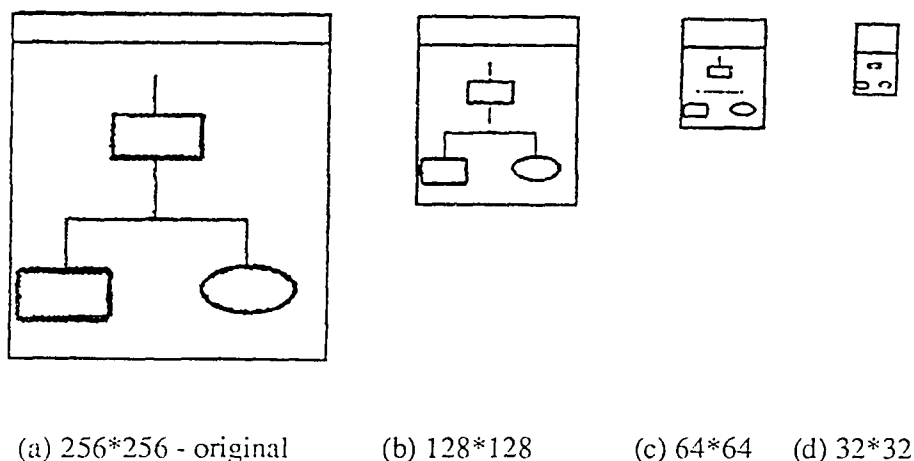


Figure 2.6 - Image Abstraction (line width = 8)

2.3.2 Image Blurring

All pixels in a sharp image contain only the local information of the corresponding symbol. Image blurring is a mechanism of blending influence of neighboring pixels into a central pixel. Any bounded symbol is usually an aggregation of its component pixels. Through blurring, an aggregation of close-together pixels will generate a local intensity maximum around the geographical center of that symbol and preclude other unrelated single lines and aggregation of pixels due to either remoteness or lack of enough black pixels. This is very much like the zoom-out effect of a picture taken by a camera. Some significant objects will become groups of vague objects after zooming out. Under appropriate de-focusing, this local peak will be situated inside the symbol and influence from pixels not belonging to that symbol will not be strong enough to drag this peak out of the symbol boundaries. Thus, by checking local maximum positions, the symbols can be approximately located. The only restriction is that the distance between symbols should not be smaller than the symbol's size so that the influence from pixels of the involved symbol is ensured to exceed that of other pixels.

A carefully selected defocus distance is critical in generating a blurred image whereby each local intensity maximum represents a symbol in the original image. The defocus distance decides the distance within which each pixel will be influenced by every other pixels. In case of a small defocus distance, only the pixels that are close enough will be influenced by each other and pixels at the two ends of a symbol is still too far away to exert any influence. When the defocus distance is too large, pixels from one symbol may receive greater influence from extrinsic pixels than their intrinsic pixels resulting in no local maximum being generated. Take a sharp image of a single circle for example (Figure 2.7), if the defocus distance is too short, the resulting image will contain a group of local maximums inside the circle (Figure 2.7(a)). If the defocus distance is appropriate, a local

maximum will appear inside the circle (Figure 2.7(b)). Obviously, the defocus distance depends entirely on the symbol's size. However, it is not very depending on the exact size. A local maximum will still appear in the example image if one uses a greater defocus distance, for instance, as shown in Figure 2.7(c).

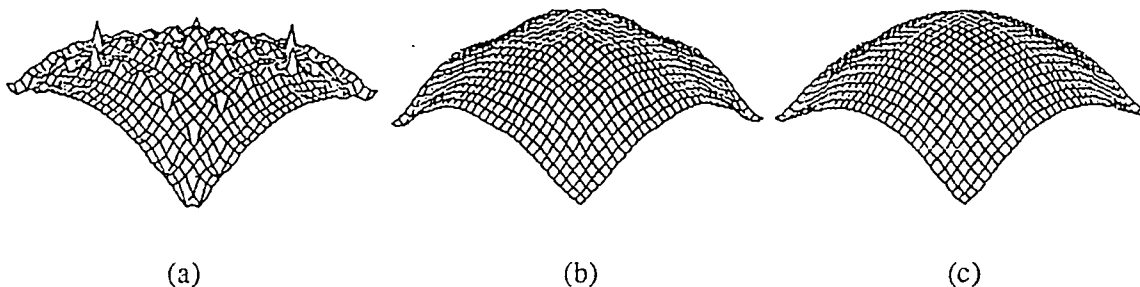


Figure 2.7 - Image Blurring (a) $f = 0.5s$ (b) $f = 0.8s$ (c) $f = s$ (where f is the defocus distance and s is symbol size).

The blurred image is generated through a two-dimensional Gaussian filter. Each pixel in a sharp image will influence other pixels by

$$i(x, y) = \sum_{(x', y') \neq (x, y)} \frac{1}{2\pi f^2} e^{-\frac{(x-x')^2 + (y-y')^2}{2f^2}} \quad (2.1)$$

where $i(x,y)$ is the intensity of pixel (x,y) , and f is the blurring coefficient or defocus distance. Experiments show that for symbol size s , a defocus distance of $0.8 \times s$ will be appropriate.

Theoretically, each pixel in the blurred image is affected by any other pixel. However, since the magnitude of influence decrease exponentially with respect to the distance, the computational time is reduced by calculating only influences from pixels that are close enough to each other. From the experiments, it is clearly that the peak position

will not be influenced by pixels 3 times the symbol's size away. Thus, for an $n \times n$ image, the amount of computation for the Gaussian function is reduced from $O(n^4)$ to $O(sn^2)$, where s is the symbol size.

In an engineering drawing, symbols may not be of the same size. For instance, CAP and GROUND are usually drawn in smaller shapes than AND gates or FLIP-FLOPs. Although the appropriate defocus distance is not very sensitive to the symbol's size, a single defocus distance is not sufficient for processing raw images with vastly different symbol sizes. In this case, a series of blurring is being carried out. As mentioned above, if a defocus distance is not large enough, it will generate many local maximums around the symbol. In other words, small symbols will not be detected if the defocus distance is too large. In lieu of the above, the blurring process is started with the largest defocus distance, locating the local maximum for larger symbols, after which it is segmented using a moving window as described in the following section. Next, the image is blurred using the next largest defocus distance. Any local peak within the already segmented image area will be ignored. The remaining local peaks correspond to the symbols with the next largest size. These steps are repeated until the smallest symbols are completely segmented.

2.3.3 Symbol Segmentation

After the symbol locations have been located, a window as large the symbol is displaced around each peak of the original image until the entire symbol is within the window (Figure 2.8). The window that captures an entire symbol must contain the maximum number of black pixels among all windows covering the peak. The requirement that the average symbol distance should exceed the symbol size works here to ensure the correct segmentation.

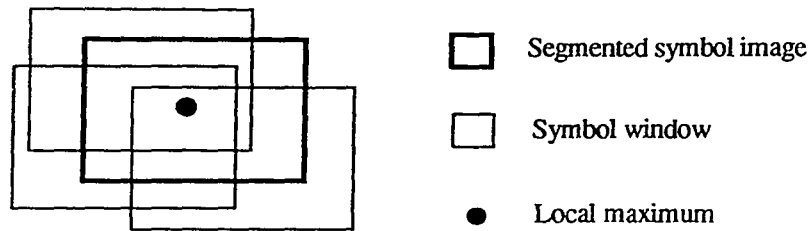


Figure 2.8 - Symbol Segmentation Around the Local Maximum after Blurring

Image blurring is performed starting from the largest symbol even if there are multiple-sized symbols. Windows with the corresponding sizes are used to segment symbols of various sizes, i.e. smaller windows are used to move around the local minimum of blurred image with smaller defocus distance. A segmented image part is ignored if it is within the area that has already been segmented in previous steps.

2.3.4 Parallelization of the Overall Segmentation Process

Each step of the proposed segmentation algorithm can be implemented as a parallel algorithm in a multi-processor computer. Calculations in the abstraction step and blurring step are being performed for each pixel sequentially. These calculations have no relative data dependencies among different pixels and thus are suitable for parallelizing individually. Since the raw image is two-dimensional, it can be divided evenly according to the number of processors available (data parallelization), shown in figure 2.9. Let the original image size is $M \times M$ and there is $(N+1)$ processors, the size of sub-image for each slave processor will be $((M/\sqrt{N})+2) \times ((M/\sqrt{N})+2)$. After each iteration, the image size will become the one fourth of the original one. In actual implementation, it is possible to load entire image to each processor initially and pass the origin and the size of image to be executed to the slave processor, which can reduce the data communication cost.

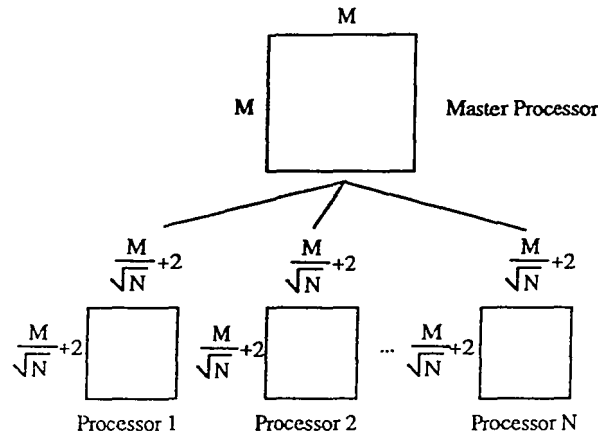


Figure 2.9 - Image Abstraction on (N+1) Processors for M*M Line-drawing Image

Similar to the image abstraction process, during image blurring process, each slave processor will receive the coordinates of an origin and the size of the sub-image to be blurred. As mentioned in section 2.3.2, the size of each sub-image is always 3 times the symbol size. Parallelization for the third step can be performed at an even higher level. For example, each processor can be responsible for a local maximum (symbol location). Since the data (image) only needs to be downloaded to the slave process once, a completion message should be acknowledged at the end of each iteration. The central (master) processor plays an important role in issuing command, passing and receiving parameters to and from every slave processor. The parallelization of the overall segmentation process is achieved.

2.4 EXPERIMENTAL RESULTS

Several examples of segmenting logic gates from electrical engineering drawings are shown in Figure 2.10 in next page. The experiments were done on SUN SPARC¹

¹ Product Trademark of SUN microsystems Inc.

workstations and the images were captured by HP Scanjet² using a Black/White mode at a resolution of 150 dpi (dots per inch) in TIFF (Tagged Image File Format) representation. All test images have a size of 256×256. Figure 2.10(a) is an example with three close-loop symbols. A dot which represents the local maximum after image blurring process is shown at near center of each symbol. Figure 2.10(b) is an example with four open-loop symbols. The result of the upper-left symbol has been shifted right a little bit because the line width of the right vertical line is much greater than the average line width, 3. Figure 2.10(c) is an example that includes symbols of different sizes and both open- and close-loop symbols. Figure 2.10(d) shows one error segmentation on the two crossing lines and one extra segmentation at the center of drawing because the pixels are distributed on balance at two sides in those regions. The error segmentations can later be detected and rejected during the symbol recognition process, which will be introduced in the next chapter. These experimental examples show that the new segmentation approach can correctly segment either open- or close-loop symbols of different sizes.

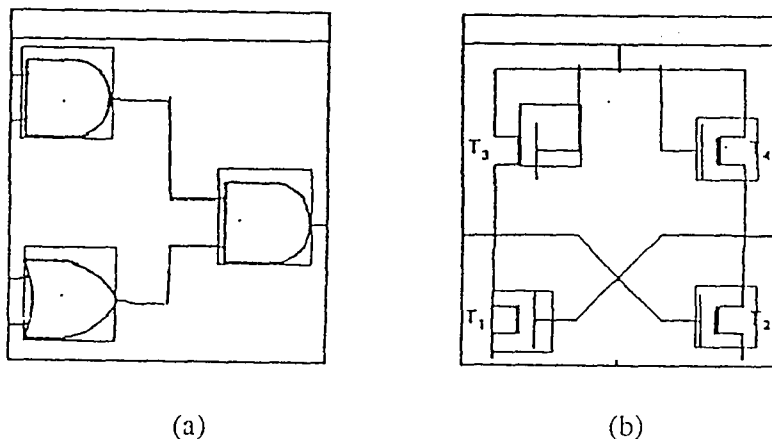


Figure 2.10-- Experimental Results on Symbol Segmentation

² Product Trademark of Hewlett-Packard Inc.

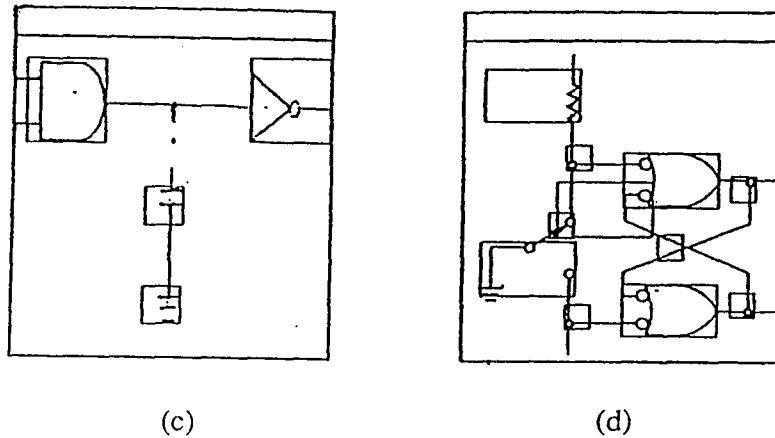


Figure 2.10 - Experimental Results on Symbol Segmentation (contd.)

2.5 SUMMARY

A new symbol segmentation method based on image abstraction and blurring is developed to meet the needs of an automated electrical symbol recognition system. It is observed that most of the current approaches use thinning or vectorization as required pre-processing steps followed by sequentially skeleton tracing but the proposed approach can be implemented in parallel machines. The experiments further proved that the blurring-based algorithm works well for both open- and close-loop symbols with different sizes. It is a global and fuzzy (human-like) way to detect the locations and sizes of symbols. Due to the blurring effect, noise of the given line drawing image will be filtered since they are usually not aggregated. Therefore, it is less sensitive to the noise compared to other approaches. The only drawback of this approach is that the segmentation results are affected by the density of the surrounding connecting lines. As a result, it may identify more symbols than the actual number in some cases. However, this drawback can be overcome by the following symbol recognition process, since the erroneous segmentations will not be recognized as pre-defined symbols in the encoded symbol model. Thus, the feasibility and advantages of segmenting symbols by image abstraction and blurring has

been proved and demonstrated. In addition, an extension to the feature detection of image or text segmentation for any paper documents is also proposed in this research. The following illustration in Figure 2.11 shows some preliminary results of applying this three-stage symbol segmentation method to text and image (picture) data.

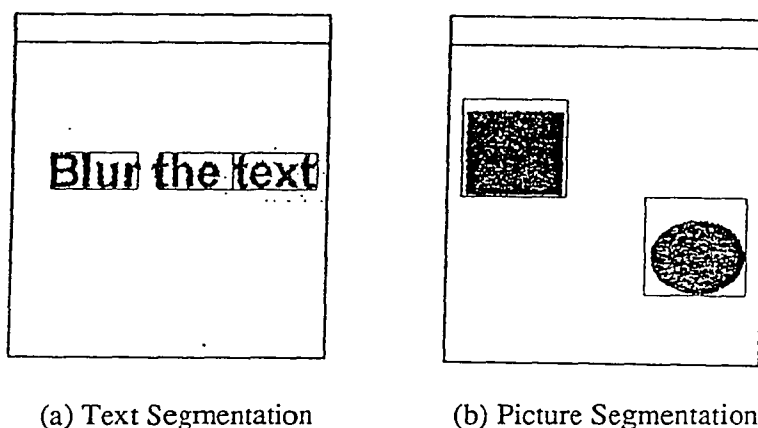


Figure 2.11 - Some Test Results of the Text and Picture Segmentations

Image segmentation is the most fundamental task for understanding the contents of the given image. At present, there is no general technique for segmenting text, symbols, graphics primitives, and pictures all together since most of current researches are emphasizing on the recognition of segmented images or how to compress these images so as to reduce the storage size and communication cost. In summary, based upon the image abstraction and image blurring techniques, a general segmentor, shown in Figure 2.12, for various types of documents is expected to be developed in the future.

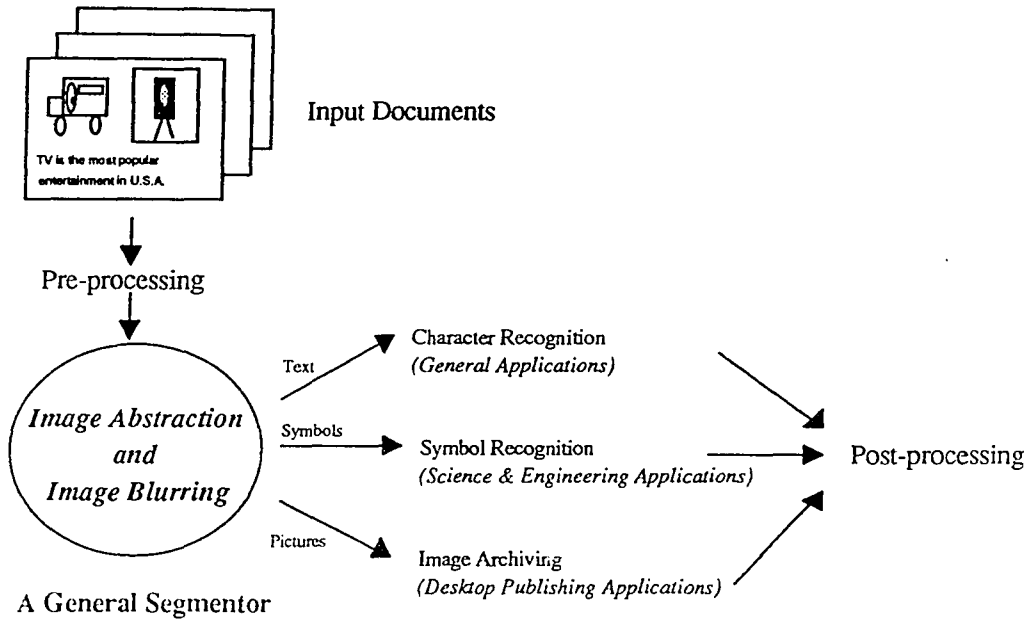


Figure 2.12 - A General Image Segmentor

CHAPTER 3

SYMBOL MODELING AND RECOGNITION

3.1 INTRODUCTION

A long time industrial need has been identified as *automatic conversion of image documents (ACID)*, which includes the process of scanning paper drawings to recognize of symbols (including characters) and lines, as well as understanding their relationships and semantics. After the relative success in character recognition (as seen in the recent commercialization of pen-based computers), automatic image document conversion is naturally the next frontier in the total *image intelligence* (feature extraction, recognition and understanding) arena.

The majority of current solutions for ACID involve vectorization, thinning, edge detection, segmentation and recognition based on pre-defined object/symbol models . The drawbacks of these methods include intensive human involvement to encode symbol models, the need of special hardware (e.g. vectorizer) to speed up the conversion process and the lack of adaptability with increasing level of image complexity or number of symbols. The goals of this work are to develop a technique and the associated prototype system to facilitate symbol model encoding as well as to allow symbol set update and modification with minimal human recoding.

Neural network (NN) techniques have been developed to mimic human biological nervous and learning systems. A NN is composed of numerous simple processing elements called neurons inter-connected via weighted links. The strength of each link is updated through a series of iterative training steps that present training samples to the neural networks one by one until it converges (error is minimized). After training is

completed, the network produces the desired (classified) outputs via a simple parallel calculation done by every neuron, thus achieving high computational rates. Moreover, NN provides a much higher degree of robustness and adaptation than conventional classifiers. Besides, systems based on the neural network approach can tolerate input noise and be adapted to a larger application domain without any major modification.

Neural networks typically operate in two modes (phases): *learning* and *recalling*. The recalling mode maps an input data to an corresponding output data. The mapping is one to one and there is no restriction for the types of inputs and outputs. Basically, in most of neural networks, the recalling phase can be performed in constant time. On the other hand, the learning mode is much more complicated than the recalling model. The learning phase converges or stops until the outputs of two consecutive time stages are the same or their difference is less than a pre-defined value. Both operations are highly parallel and neurons communicate each other locally.

Several neural network models for solving pattern recognition problems have been proposed and applied successfully [ChKi92][KhLu88]. The major advantage of NN approaches is the automatic model extraction from training data and adaptivity to new symbols. This in turn can significantly reduce human involvement.

This chapter presents a hierarchical neural network approach for the automatic conversion of image documents. Specifically, a Symbol Recognition System (SRS) prototype for automatic processing of electrical engineering drawings will be described. This approach achieves a significant reduction of human involvement in the symbol model encoding and recognition processes, in contrast to traditional approaches based on thinning, line tracing and other geometrical feature extraction techniques.

A set of image intensity moments are used as features which are naturally more easily extracted and evaluated by machines. These features also demonstrate a higher level of cognitive invariances like scaling, translation and rotation. This hierarchical

approach have proven to have a *faster* and *more accurate* capability for model encoding and recognition. The test results from hand-drawn images using templates show that the hierarchical neural network can easily achieve a recognition rate of 98.5% on training symbols and 89% on new test symbols (not encountered during training). The analysis of the number of training cycles and the width of the hidden layer are provided. The hierarchical organization of the neural network also facilitates *incremental extension* and *minimum disturbance modification* to the already encoded model. Thus, this approach possess the properties of allowing adaptability to other engineering drawing domains as well as scalability to complex and varying real-world applications.

The remaining sections of this chapter is organized as follows. In section 3.3, the pre-processing of the segmented image is described. Section 3.4 discusses the feature extraction using geometric moments. The hierarchical Multi-Layer Perceptrons (MLP) with the Back Propagation (BP) learning model is covered in section 3.5. Section 3.6 presents the experimental results on both monolithic and hierarchical neural networks. The summary and future work of this research are given in the section 3.6.

3.2 PROBLEM DEFINITION AND RELATED WORK

Symbols and characters are two major information to be recognized in an engineering line-drawing. The character recognition is one of mature sub-topics in drawing interpretation research area and many commercial products are available in market. However, the progress of symbol recognition research has not shown apperantly in recent year. Since (electrical) engineering symbols are usually connected, have various sizes and can be located anywhere and in any orientation, symbol recognition is much more difficult than character recognition.

Most of previous solutions of identifying symbol types from the segmented symbol regions are based on structural pattern analysis method that analyzes the segmented line structures by either decision-tree [OKMT88], graph-matching [Bunk82][GrSS85][Lee92][KiSK93], or knowledge-based [ToHL84] approaches. These previous solutions involve vectorization, thinning, edge detection, segmentation and recognition based on pre-defined object/symbol models . Their decision-tree or rules of symbol interpretation are encoded and could not be adapted to new symbols or new applications. Since open-loop symbols can not be represented as a graph structure, the most of above approaches are not able to recognize or need special mechanism to handle those open-loop symbols.

3.3 FEATURE GENERATION

Selecting representative feature from an image is a critical step in the pattern recognition process since each neuron in the classifier can only see a portion of the whole image. Every (segmented) symbol should have the same number of features that are invariant to their orientation, size and location in the entire drawing image. The moment-invariant approach, which is invariant to rotation, scaling, and translation, was first introduced by Hu [Hu62]. Applications of this approach to pattern recognition such as aircraft identification [DuBM77], 3D shape analysis [RPAK88] and character recognition [KhLu88], etc have been met with significant success. In this research, six moment invariants are selected as features to classify 23 logic-gate symbols from schematic drawings. To ensure the stability of these moment features, some pre-processing steps such as autocropping, centralization and normalization will be needed before feature extraction.

3.3.1 Image Pre-processing

The image pre-processing step performs three basic functions on the segmented symbol images: *autocropping*, *centering* and *normalization*. This pre-processing step is necessary in order for the bitmaps to be presented to the feature extraction module in a consistent way, invariant to the position and size. The pre-processed image is then centered onto a 64 by 64 pixels window.

The autocropping pre-processing can eliminate the potential errors arising from user cropping, i.e. user may crop symbols with windows of different sizes. Under such circumstances, unstable scaling may occur. The autocropper finds the real boundary of an image by searching for extreme black pixels in the following positions: the right-most, the left-most, the upper-most, and the bottom-most. To achieve rotational invariance as defined in the moment extraction function, the symbol must be located at the center of the window. The centroid of a symbol is calculated by taking the average of both x and y coordinates of all black pixels in the image window. After the symbol is centered, a scaling operation is performed if the size of symbol is greater than the window size. The larger scaling factor of both sides of the symbol is selected to maintain a correct aspect ratio and thus prevent shape distortion. And also if the symbol is smaller than the window, no scaling is performed to avoid scalar distortion.

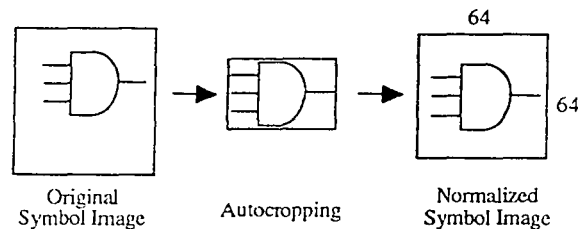


Figure 3.1 - Image Preprocessing

3.3.2 Invariant Feature Extraction

Electrical engineering drawing symbols may have different sizes and orientations and their positions are usually distributed randomly in the entire drawing. To recognize the symbols independently of their position, orientation and size, six invariant moment features are extracted from each symbol image as inputs to the hierarchical neural network. The invariance and domain-independent properties of these six invariant moments are the principal reasons for using them as features in the symbol recognition application.

The generic moment function for a given two-dimensional discrete image, $f(x,y)$ takes on the following form [TeCh88]:

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} x^p y^q f(x,y), \quad \text{where } p, q = 0, 1, 2, \dots \quad (3.1)$$

where $f(x,y)$ is the intensity and $x,y = 0, \dots, M-1$.

To make an image invariant to translation, the moments should be mapped into the centroid of the image. The central moments can then be calculated by the following equation:

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} (x - \bar{x})^p (y - \bar{y})^q f(x,y), \quad \text{where } \bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}} \quad (3.2)$$

Central moments are then normalized for size invariance by defining

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^\gamma}, \quad \text{where } \gamma = \frac{p+q}{2} + 1 \quad (3.3)$$

The centered, normalized and rotational invariant moments are then derived from a set of non-linear functions defined as follows:

$$\begin{aligned}
\phi_1 &= \eta_{20} + \eta_{02} \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
\phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - (3\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
\phi_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21}).
\end{aligned} \tag{3.4}$$

The logarithmic values, i.e. $\log|\phi_i|$, where $i=1 \dots 6$, of these moments are used to scale up the small numerical values. In this work, only six moments are used primarily due to the excellent results (showing their sufficiency) obtained by Khotanzad et. al. [KhLu88] in character recognition.

3.4 HIERARCHICAL NEURAL CLASSIFIER

Among the various neural network classifiers, the multi-layer perceptron (MLP) classifier using back propagation (BP) learning algorithm has been successfully applied to many pattern recognition problems especially in Optical Character Recognition [ChKi92, KhLu88]. However, in many practical pattern recognition problems, single neural network classifier such as MLP with BP learning model does not converge to its solution state. Even if the network converges, the time required for convergence may be too long for practical usage. Therefore, in this new prototypical system, a hierarchical

neural network with two-stage MLPs under BP learning is proposed to improve the convergence time as well as the recognition rate over traditional monolithic MLP. In this model the symbols are first classified into four major classes, after that each class is assigned a specialized network to perform further classification and fine-tuning.

A neural network can be considered as a mapping function, F , which maps input I into the corresponding output O ($F: I \rightarrow O$). A neural network classifier maps the feature space into a set of output classes. Recently, many successful applications using neural networks, especially MLP trained with BP algorithm [KhLu88], as the classifier in either image or speech recognition problems, have been implemented in place of traditional classifiers.

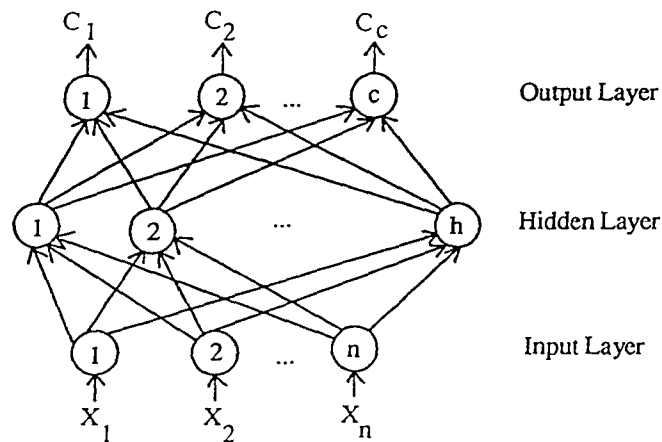


Figure 3.2 - A Two-layer Perceptron Network

Figure 3.2 shows a two-layer perceptron with n input units, h hidden units and c output units. All input-layer units are fully connected to the hidden layer units, which in turn fully connected to the output-layer units.

During learning, an input vector (in feature space) $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is applied to the input layer. First, the weighted input summing into each hidden unit i is computed as:

$$I_i = \sum_{j=1}^n w_{ij} x_j \quad (3.5)$$

where w_{ij} is the weight connecting input unit j and hidden unit i . Next, this input value is passed through the activation function, a *sigmoid* function, such as:

$$f(I_i) = \frac{1}{(1 + e^{-I_i})} \quad (3.6)$$

After this process, the output of each hidden unit is obtained. The output of each output unit can be calculated in a similar way. The output unit with the highest output value represents the input vector class.

The back propagation algorithm uses a *generalized delta rule* to minimize the errors with respect to a series of examples. The generalized delta rule specifies the change in a given connection weight as: $\Delta w_{ij} = \beta E f(I)$. Here E is the error for this unit, β is the learning constant parameter between zero and one and $f(I)$ is the input to the unit. The net error in each hidden-layer unit is thus the weighted sum of the error contributions from each of the output-layer unit. The error computation for both output-layer units and hidden-layer units are as follows:

$$E_j^{output} = y_j^{desired} - y_j^{actual} \quad \text{and}$$

$$E_i^{hidden} = \frac{df(I_i^{hidden})}{dI} \sum_{j=1}^n (w_{ij} E_j^{output}) \quad (3.7)$$

where $df(\cdot)/dI$ is the derivative of the sigmoid function defined in (3.6).

In summary, BP algorithm performs gradient descent in the weighted space until the total error between the desired and actual outputs of all nodes is stable and reaches a minimum. It has been shown that a two-hidden-layer MLP is capable of forming any arbitrarily complex decision region [RuHW86]. In addition, Kolmogorov's theorem [Kolm63] suggests that the required number of hidden units is $2n+1$, where n is the number of input units. However, in practice, more neurons are required to obtain reasonable performance, especially when the number of output classes is large. This will be discussed in the next section.

Due to the previous experience in designing a practical neural network for the pattern recognition problem [ChKi92], the conventional single-level neural network classifier known as *monolithic neural network* has poor convergence performance. Even if it converges, the time required is usually too long and time-consuming for practical applications. Therefore, a two-level hierarchical neural network (Figure 3.3) to train and test the invariant moments extracted from the raw image has been developed based upon various experiments. It is obviously show that this hierarchical neural network performs better than the traditional monolithic network. This hierarchical architecture simplifies the learning task of each neuron and each stage can be trained concurrently and independently. On the other hand, the number of output classes of each sub-network is smaller, thereby decreasing the size of hidden layers and shortening the training time.

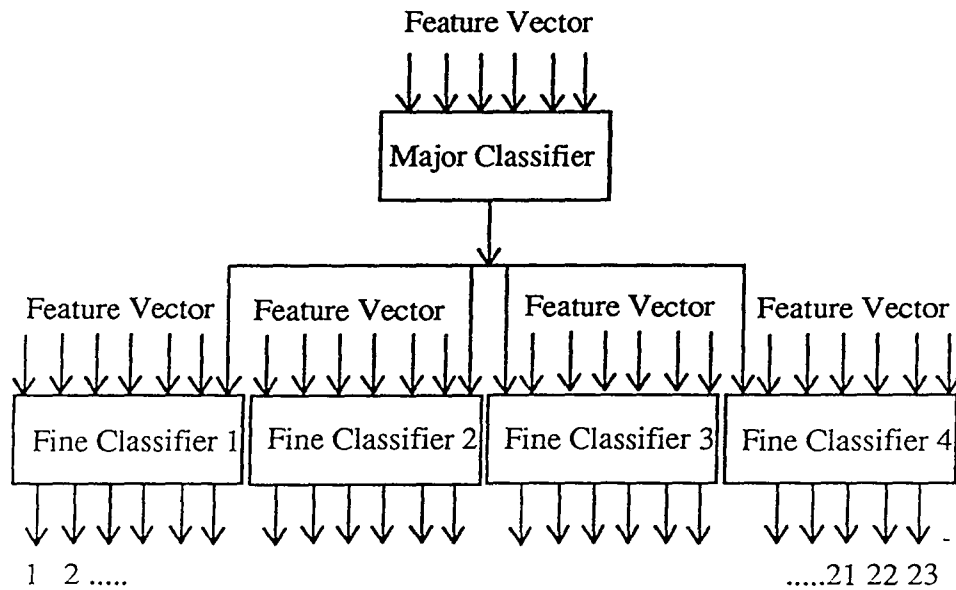


Figure 3.3 - Hierarchical Neural Network Classifier

The recall process also works faster for the same reason and the modular design makes system maintenance easier. After considerable training, if some new symbols need to be added, the entire network need not be retrained from scratch. Only the top level classifier among the existing modules requires retraining. The concepts learned by the lower level classifiers can be left as is. Likewise, if some previously learned model is required to be retracted, only the involved modules along the hierarchy need to be retrained. In the same way, the region of generalization/specification on individual components can be selectively adjusted with greater flexibility. Such a NN architecture with reusable components becomes increasingly critical in resource savings situation (computer training time, human expert effort, etc), especially with increasing complexity of target applications.

3.5 EXPERIMENTAL RESULTS

A prototype that has been built is called the Symbolic Recognition System (SRS). In essence, the system takes a scanned electrical drawing as input and through user interaction (comprises primarily manual segmentation for this SRS prototype), provides recognition results on the user's display. The SRS architecture is shown in Figure 3.4. The user interface is a graphical environment based upon X-windows on the Sun workstation. The user is provided with several pull-down menu options to control operations of the SRS application such as reading image file, selecting the symbols and recognition settings, etc. The image segmentation module is designed to automatically perform both symbol segmentation and pre-processing.

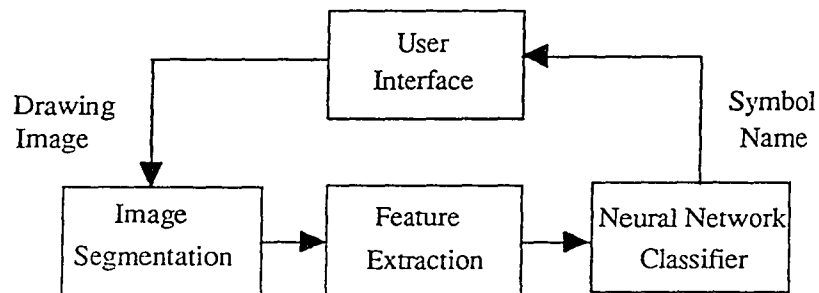


Figure 3.4 - SRS Basic Architecture

At the present stage, the segmentation step is performed under the user's supervision. The feature extraction module performs a set of non-linear mathematical operations on the pre-processed bits in a 64 by 64 window. These operations generate a vector of 6 fractional numbers, referred to as the invariant moments of the bits. These invariant moments are expected to be unique to each symbol type, and these "*symbolic signatures*" form the basis for the neural network based recognition. The classification module takes the invariant moments for the given scanned bits and identifies the closest

matching symbol. The classified symbol and other analytical data are then passed onto the user interface module for displaying.

Some results of the experiments are presented and discussed below. Figure 3.5 shows the dynamic convergence behavior of the network as training progresses for the symbol set of Figure 3.7. The X-axis represents the training progress and the Y-axis represents the network detection efficiency in terms of error-count. The lower curve shows how the error can be reduced when tested with the images used during training. The upper curve shows the performance on a test set not used in the training. Clearly, the training set detection rate is better. Another important observation is that training error rate keeps on improving with more and more training, although at a diminishing rate. Nevertheless, performance with the unseen test set stabilizes quite early at about 2000 cycles.

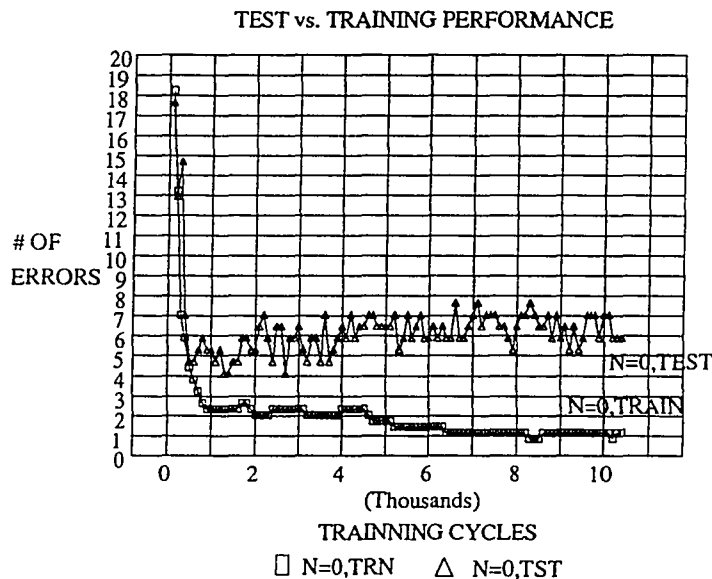


Figure 3.5 - The Detection Rates for Both Training and Test Sets

Another important factor in BP training is determining the optimal number of hidden layers. Below is an experiment where the performance of the major classifier is

monitored with varying hidden layer width. The width of the hidden layer can greatly influence the training/recovery time. Figure 3.6 reveals that 20 is approximately the best width for the hidden layer in this application. The error rate increased when there was less than 20 hidden units being used. On the contrary, the detection efficiency dropped with approximately 30 hidden units. The reason of this behavior can be informally explained in terms of the generalization capability of the BP learning. Conceptually, each of the hidden units can be visualized to represent abstract intermediate concepts from the image. If the number of hidden units is small and the images are sufficiently complex, the performance suffers because of the lack of intermediate concept storage space. Conversely, if too many hidden units are employed, then the network tends to memorize the image instead of learning the general concepts. As a result, when the network is faced with unseen images, it fails to demonstrate the expected detection capability.

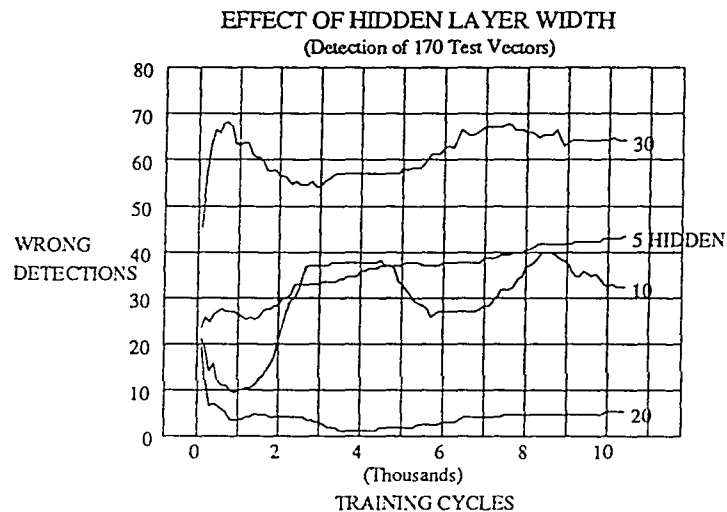


Figure 3.6 - Hidden Layer Width Selection

The MLP with BP learning algorithm was used as the basis for symbol classification due to its good track record capability to classify a input set into a set of desired outputs. This module was implemented as a two-level hierarchy with one major

classifier and four minor classifiers as shown in Figure 3.3. Each minor classifier had approximately 6 symbol outputs (for a total 23 outputs) in Figure 3.7. This hierarchical structure contributes to the efficient training time and slightly better classification accuracy over the monolithic structure. The architecture for the monolithic neural network has 6 inputs, 20 hidden units and 23 output units. Each of the input unit corresponds to an element of the feature vector and each of the output unit corresponds to a class. The internal structures of the major and minor classifiers are just like the monolithic one, except in the number of output units.

For the symbol set shown in Figure 3.7, the SRS prototype was trained with 340 input patterns taken from a set of 34 symbols grouped into 23 classes (some symbols, e.g. AND and AND-L, are considered to be from the same class). Each set also includes rotated symbols. The training constant (learning rate) was varied from 0.5 to 0.05 with a corresponding momentum of 0.1. 170 additional input symbols were used for the testing phase (after training) and the following recognition results were achieved:

- Correct Recognition: 85.9% (Output symbol class matches scanned test data)
- Class Rejection: 3.5% (Low confidence by all class)
- Incorrect Recognition 10.6% (Recognized class is a mismatch)











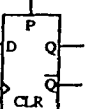
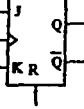





















Group 1 (6 classes)	Group 2 (6 classes)	Group 3 (6 classes)	Group 4 (5 classes)
 AND  OR	 PNP  NPN	 CAP  XTAL	 RES  RES_V
 NAND  NOR	 DFF  JKFF	 CAP_L  GND	 FUSE  SWTCH
 AND3  OR3	 INV  MOSFET_N	 OPAMP  XFRMR	 DIODE  IND
 AND_L  OR_L	 INV_L  MOSFET_P	 CHGND  BAT	 DIODE_L
 XOR  XNOR			

Figure 3.7 - Symbols Used in SRS system (34 symbols, 23 classes and 4 groups)

In the above experiment, a modified interpretations of the term "recognition" is used. In the traditional sense, recognition means the maximum relative response of each class identifier. In this situation, the output producing the maximum response is used as the class of the input. However, this interpretation runs the risks of committing "blunders". Just winning the response bid is not a good enough indication of the classification. If the winning bid is too low, then it is safer to label the case as non-recognition or "rejection", meaning that the network is confused. Therefore, another more cautious interpretation is used, in which if the output showing maximum relative response falls below a threshold then it is classified as "rejection" rather than "recognition".

After the recognition, the results were divided into correct and incorrect categories. In table 3.1, the column "Error by MAX" shows the result according to the first criteria. On the other hand, the column, "Error by Swing" refers to the error according to the second and more restrictive criteria. Since the values 0.9 and 0.1 are used to represent class and non-class respectively, a maximum swing of 0.4 is allowed to validate a decision. Thus, any value between 0.0 and 0.5 is considered as 0.0 and any

value between 0.5 and 1.0 is considered as 1.0. When one and only one output response is between 0.5 and 1.0 range, it is considered as a successful recognition. The "Error by Swing" refers to the recognition error according to this criteria. The ability to detect the "confusion" is important from a decision-making point of view. The successful recognition rate on the 340 training input data and 170 test input data are 98.5% and 89% respectively. For the monolithic structure, the recognition rate is 80% with test symbols and 97.5% with the training symbols.

Table 3.1 - Component Error Table

	Patterns	Layer	Error by SWING	Error by MAX	MSE
Training	340	Major	3	3	7.39
		Fine1	0	0	4.04
		Fine2	0	0	0.68
		Fine3	0	0	0.32
		Fine4	0	0	0.36
		Total	3	3	x
Test	170	Major	13	7	10.88
		Fine1	13	5	10.59
		Fine2	5	4	4.32
		Fine3	2	2	2.33
		Fine4	5	5	5.73
		Total	38	23	x

3.6 SUMMARY

A successful attempt is shown in this chapter to increase the automation in model generation for recognizing electrical design documents. A neural-network-based symbol recognition system has been developed and demonstrated on an application of electrical engineering drawings. Six invariant moments were selected as features and a hierarchical MLP with BP learning algorithm was implemented in this system. The experimental

results show that the correction recognition rate are 98.5% and 89% on training and test set of hand drawn symbols, respectively. This result demonstrates the effectiveness of the hierarchical classifier. From the experiments shown in this research, it is clear that the moment features and the hierarchical neural network for symbol modeling and recognition have met *the ultimate goals: faster symbol model development, speedier recognition, more robustness and adaptability and less human involvement.*

Experience from simulating the monolithic neural classifier reveals that the addition of some high level human judgment can significantly improve modeling and recognition accuracy. Generally, the complexity of monolithic classifier tends to increase the training time super-linearly as well as decrease the recognition accuracy as it grows in size. On the other hand, hierarchical classifier requires increased human intervention in the form of appriori-class assignments of the symbols on the basis of similarity. In this case, a higher-level human assistance has been utilized to improve the practical limits of neural networks. The human expert is required to assign classification labels to the symbols according to their perceptual similarity. This intervention is done only one time, and is not very tedious. However, as a payoff this assistance greatly reduces the burden on the neural classifier both in terms of training time and recognition accuracy.

The modularity of the hierarchical classifier also facilitates the maintenance of the generated model. A new class can be added by training the new minor classifier module and retraining the major classifier module. Moreover, an existing class can be modified without disturbing previously learned unrelated modules. In addition to these maintenance advantages, the training of the minor modules can be performed concurrently. Therefore, hierarchical classifier system offers a promising means for flexible and reliable technology in symbol recognition with minimum human involvement. The combination of the symbol segmentation technique: image abstraction and blurring approach, mentioned in chapter 2 and the result of this work holds promise for achieving

the automatic conversion of image documents in both symbol segmentation, modeling and recognition processes.

PART II

3D LINE-DRAWING INTERPRETATION

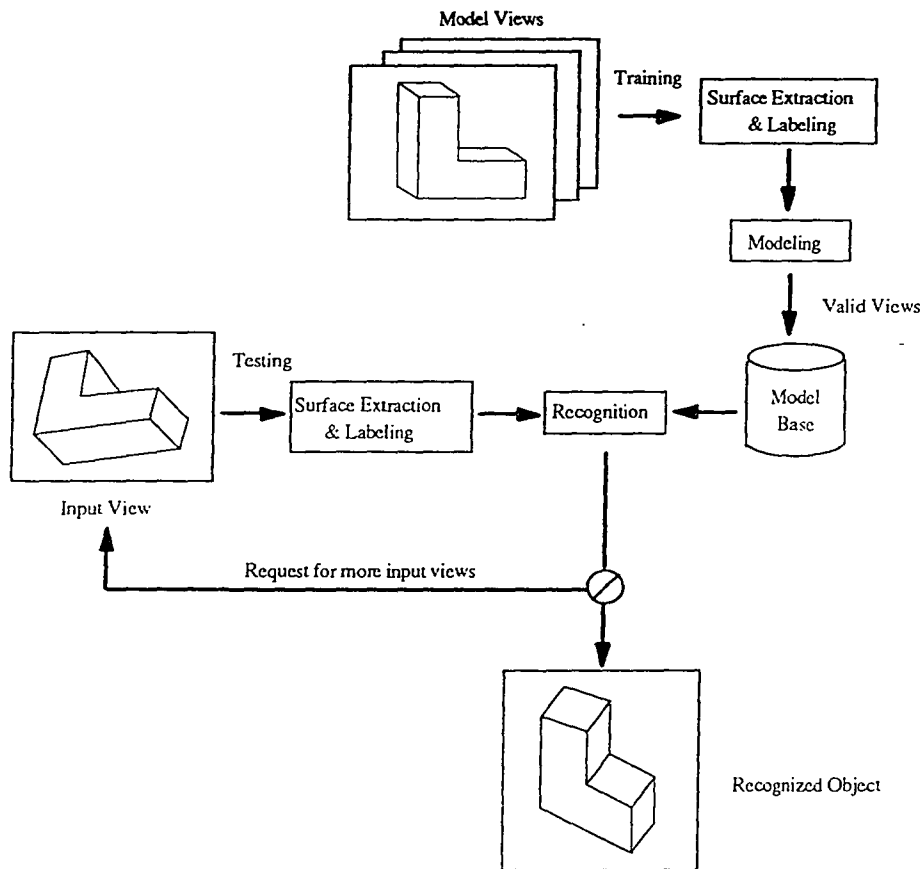


Figure II.1 - A Complete 3D Line-drawing Interpretation System Diagram

For the purpose of rapid recognition, humans represent a 3D object by a set of characteristic views or aspects, i.e. by a set of **commonly occurring** 2D projections [Rose87].

For the purpose of reliable recognition, humans perceive a 3D object usually by looking for a set of **different but informative views** from different viewpoints [CheY94].

CHAPTER 4

SURFACE EXTRACTION ALGORITHM

4.1 INTRODUCTION

A 3D object can be characterized by its line drawings generated from the intensity images of object in 3D space. Junctions in the given line-drawing correspond to the vertices of the 3D object whereas line segments are the projections of the 3D edges. Junctions and line segments form an edge map of the projected image and can be obtained by the various popular edge-detection algorithms available in literature. [Cann86] Due to the noise from the viewing environment or input devices, the line-drawing may not be perfect. In other words, missing lines, displaced junctions and spurs can often be found in the low-level image pre-processing stage. Since some heuristic search algorithms [GuHu85][Huan93] for obtaining perfect line-drawings from noisy images have been already proposed, the line-drawing-extraction subject will not cover in this research. The surface conveys significant information and is one of the essential features in characterizing a 3D line-drawing. Thus, the extraction of surfaces is of practical interest for object modeling and matcing.

In this chapter, a novel surface extraction algorithm with linear-time computational complexity will be introduced. This algorithm, called polygon-division-based algorithm, works on line-drawings projected by trihedral objects. A trihedral object is a polyhedron which has exactly three plane surfaces meeting at each vertex. This efficient algorithm, based on a polygon-division procedure, divides the larger polygon (which is not an elementary polygon) into two smaller polygons (which may or may not be elementary

polygons) at each iteration until no more division can be made. An elementary polygon is a planar surface with no interior edges and all line segments located at the boundary.

The rest of this chapter is organized as follows. A formal problem definition and some previous approaches are given in section 4.2. The polygon-division-based algorithm and the corresponding complexity analysis will be described extensively in section 4.3. In addition, an experimental result to demonstrate the linear-time performance of this algorithm will also be shown in that section. Finally, the significance of this work and the possible extensions will be discussed in the summary section.

4.2 PROBLEM DEFINITION AND RELATED WORK

4.2.1 Surface Extraction Problem

The problem need to be solved in this chapter is to automatically extract the surfaces embedded in the line-drawings of trihedral objects. The inputs to this problem are a set of line segments, of which each is a portion of a line having two endpoints on junctions. Each *junction* has a corresponding data structure comprising an index, a list of lines connected to it, and its x-y coordinates.

A *polygon* is a closed plane figure formed by three or more connected line segments. A polygon is simple [Sham78] if and only if 1) no non-consecutive line segments intersect; 2) consecutive line segments intersect only at endpoints. For simplicity, only simple polygons are considered throughout this chapter. A *surface* is a simple polygon representing a visible plane of a 3D object at a specific viewpoint. The following figure shows the relationships between junctions, line segments, and the surfaces, which are the outputs of the algorithm.

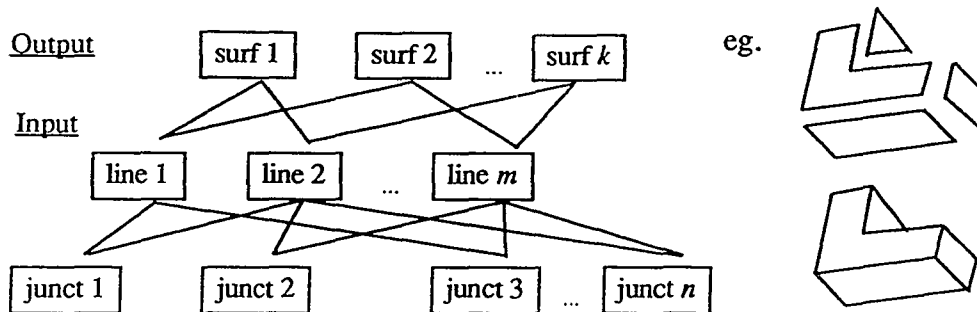


Figure 4.1 - The Surface Extraction Problem

The problem described in this research is a subset of the problem for extracting regions from a planar graph since its application domain is limited to trihedral objects. Let m be the number of line segments in the given line-drawing. Although an optimal, $O(m \log m)$ solution for general planar graph has already been proposed [JiaH93], it is believed that this new surface extraction algorithm tailored especially for trihedral objects is better because of its linear-time, $O(m)$, complexity.

4.2.2 Previous Work

Shih et al. first proposed a systolic algorithm for extracting regions from a planar graph [ShLY89]. It takes $O(m)$ computation time and uses $O(m)$ processing elements. In their approach, instead of using edges the concept of wedges is first proposed and played a fundamental role as the basic computational element. A linear time sorter is then applied to find the closed regions from the extracted wedges at the second phase.

Fan and Chang have presented a sequential algorithm by using the adjacency matrix and the minimum-positive angle seeking technique [FanC91]. No complexity analysis was given in their paper. Recently, Jiang and Bunke proposed an $O(m \log m)$ sequential algorithm based entirely on Shih's work - extracting regions from the sorted wedges. However, they have proved that this sorting-based algorithm is optimal for

extracting regions of general plane graphs and provided a complexity analysis of Fan and Chang's algorithm, which is quadratic in both time and space.

Another interesting previous approach is the "heuristically guided polygon finding algorithm" [WoKI91]. In this approach, heuristic rules are used to control the combinatorial explosion associated with unconstrained associations of junctions and triples. Physical rules are used to reject polygons which are incompatible with a single planar surface hypothesis. The algorithm works mainly for the imperfect line drawings of trihedral objects by merging two consequent line segments with the same chaining direction. It is very similar to the Fang and Chang's work. A quadratic computational complexity is expected.

4.3 POLYGON-DIVISION-BASED SURFACE EXTRACTION ALGORITHM

In this section, a new algorithm that is very efficient in extracting surfaces from line-drawings of trihedral objects will be discussed. This algorithm is based on polygon-division to divide the larger polygon (which is not an elementary polygon) into two smaller polygons (which may or may not be elementary polygons) at each iteration until no more division can be made. In the following algorithm, each polygon (surface) has associated with it an index and a list of junction indexes. Each junction has an edge list containing all edges connected to it, and a polygon list comprising all polygons sharing this junction. During the extraction process, the first extracted polygon is the outer contour of the given line-drawing and at each successive iteration, an old (divisible) polygon, if there exists any, will be deleted and two new polygons will be generated simultaneously. A step-by-step example shows how these surfaces are extracted is given in Figure 4.2. Since there are only four surfaces in the given line-drawing, four iterations are required to finish the whole dividing process.

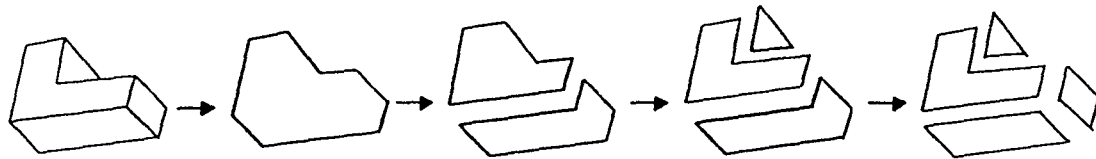


Figure 4.2 - A Step-by-step Example of Polygon-division-based Algorithm

4.3.1 Algorithms

The following is the main algorithm for extracting surfaces from a given line-drawing. The inputs to the algorithm are a set of junctions, edges and their connections. Find_Contour(·) is an algorithm for finding the outer contour of a given line-drawing in linear time, which will be explained in the following text. Store_Polygon(·) is used to store the extracted polygons into a polygon table. Test_If_Divisible(·) is a function to test if a junction has more than one untraced line segments. In other words, a polygon is divisible if such a junction could be found from its junction list. Find_Divider(·) is a procedure to find a common_edge_list that can divide the polygon into two smaller polygons. The actual division is done by Do_Divide(·) procedure. The original polygon will be deleted and the resulting polygons will be added into the polygon table by the procedure Update_Tables(·). Since each outermost WHILE loop deals with a connected component (a set of connected line segments and junctions), a hole in the object can be detected by checking for more than one connected component in which case it will be isolated and inside a surface.

```
void main(void) {
1. WHILE there exist untraced edges {
2.     polygon = Find_Contour(line_drawing);           /* Find contour. */
3.     Store_Polygon(polygon);
4.     divisible=Test_If_Divisible(&start_junction);
```

```

5.   WHILE (divisible==TRUE) {
6.       polygon_index=Find_Divider(start_junction, &common_edge_list);
7.       Do_Divide(polygon_index, common_edge_list);
8.       Update_Tables( );
9.       divisible=Test_If_Divisible(&start_junction);
    } }

```

4.3.1.1 Find the Outer-most Contour

(A) Contour-Finding Algorithm

1. Find the starting junction, SJ which has the smallest y-coordinate;
2. WHILE (not a cycle) {
3. Starting from SJ, find all of its neighboring junctions, J_s , and their corresponding connected lines, L_s ;
4. FOR all J_s
5. Compute the angle between $\langle (SJ_x, SJ_y), (\infty, SJ_y) \rangle$ and $\langle (SJ_x, SJ_y), (J_x, J_y) \rangle$;
6. Sort the neighboring junctions' angles in ascending order;
7. IF SJ is a L-type junction, we then pick the junction which has not been visited as the next SJ;
8. IF SJ is a three-line junction (W-, Y-, or T-type junction) {
9. IF the first junction is SJ, pick the second one as the next SJ;
/* Figure 4.3(a) */
10. IF the second junction is SJ, pick the third one as the next SJ;
/* Figure 4.3(b) */
11. IF the third junction is SJ, pick the first one as the next SJ;

/* Figure 4.3(c) */

}}

Note that $\langle X, Y \rangle$ represents a vector from junction X to Y . Figure 4.3 shows three different examples ordering of the edge $\langle SJ, J_2 \rangle$.

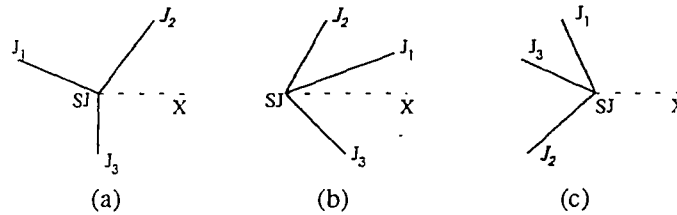


Figure 4.3 - Three Different Orders of Edge $\langle SJ, J_2 \rangle$

Hence, the junction sequences in the sorted lists shown in Figure 4.3(a), 4.3(b), and 4.3(c) are $J_2J_1J_3$, $J_1J_2J_3$, and $J_1J_3J_2$ respectively.

(B) Angle, θ_{AB} , calculation (in counterclockwise)

Let $A = \langle SJ, X \rangle = (a_1, a_2)$ and $B = \langle SJ, J_i \rangle = (b_1, b_2)$. The Cross product of A and B is defined as: $A \times B = a_1b_2 - a_2b_1$ (from A to B). The Scalar product of A and B is defined as: $A \cdot B = |A| |B| \cos\theta$. Then, we have $\theta = \cos^{-1}(A \cdot B / |A| |B|)$. If $A \times B > 0$ then $\theta_{AB} = \theta$ else $\theta_{AB} = 2\pi - \theta$.

(C) Complexity Analysis

Assuming m lines in the given line-drawing, step 1 takes $O(m)$ time to find the minimum, since each line has two endpoints. Steps 2 to 11 take $O(3)$ iterations in the worst case (visits all junctions) since each iteration needs to search at most three lines to compute and compare their angles. Thus, the complexity of this while loop is $O(m)$.

Therefore, the total complexity of the contour extraction algorithm is in the order of $O(m)$, which is proportional to the number of line segments in the given line-drawing.

4.3.1.2 Find a Divisible Polygon

This subroutine return a TRUE argument and a starting junction for the division if there exists an untraced edge in one of the junctions of the extracted polygons. Otherwise, it will return a FALSE argument.

```
int Test_If_Divisible (int *start_junction)
{
1. FOR each extracted polygon, Pi {
2.   FOR each junction, Ji, in the polygon {
3.     IF ( number of untraced edges > 0 ) {
4.       *start_junction=Ji;
5.       RETURN (TRUE);
   } } }
6. RETURN (FALSE);
}
```

4.3.1.3 Find a Divider

This subroutine extracts a Common_Edge_List (CEL) and a polygon_index of the polygon to be divided.

```
int Find_Divider(start_junction, **CEL)
{
1. Store start_junction into *CEL;
2. FOR(;;) {
```

3. Obtain one of the untraced edges, E_i , from the start_junction;
4. Obtain the other end of E_i , as the end_junction;
5. If (end_junction is not a member of *CEL) {
6. Compare polygon_lists of start_junction and end_junction.
7. IF (there is no intersection) {
8. Store end_junction into *CEL;
9. start_junction=end_junction;
10. Go to step 3;
- }
11. IF (there is one intersection) RETURN (the intersection);
12. IF (there are more than one intersection) {
13. p_index=Point_Inclusion(middle of start_junction and
 end_junction, the intersections);
14. RETURN(p_index);
- }
15. ELSE find another untraced edge;
- } }

4.3.1.4 Do Actual Division

The main functionality of this subroutine is to divide the polygon p_index into p1 and p2 according to the given Common_Edge_List (CEL). The Polygon size is n and the CEL size is $last+1$.

```
void Do_divide (p_index, CEL)
```

```
{
```

1. Obtain the junction list of polygon p_index from polygon_table;

```

2. P1=P2=NULL;
3.   FOR (i=0; n-1; i++){
4.       IF ( (Vi != CEL[0]) or Vi != CEL[last] ) {
5.           move Vi to P1;
6.       }
7.       IF ( Vi == CEL[0] ) {
8.           P1=P1 + CEL;
9.           P1=P1 + (all Vk after CEL[last] of Polygon);
10.          P2=(all Vk between CEL[0] and CEL[last] of Polygon) +
11.              reverse(CEL);
12.      }
13.      IF ( Vi == CEL[last] ) {
14.          P1=P1 + reverse(CEL);;
15.          P1=P1 + (all Vk after CEL[0] of Polygon);
16.          P2=(all Vk between CEL[last] and CEL[0] of Polygon) + CEL;
17.      } }
18. Delete polygon p_index and Store P1 and P2 into polygon_table.
19. }

```

4.3.2 Complexity Analysis and Comparison

4.3.2.1 Jiang & Bunke's Algorithm

- Input: A list of undirected edges in an arbitrary order.
- Output: A list of wedges for each extracted region.

Step	Jian & Bunke's Algorithm	Complexity
1	<ul style="list-style-type: none"> • Duplicate each undirected edge (v_i, v_h) to form two directed edges $\langle v_i, v_h \rangle$ and $\langle v_h, v_i \rangle$. 	$O(m)$
2	<ul style="list-style-type: none"> • Compute the angle θ of each directed edge $\langle v_i, v_h \rangle$ to form a list of $(\langle v_i, v_h \rangle, \theta)$. 	$O(m)$
3	<ul style="list-style-type: none"> • Sort the list into ascending order using v_i and θ as the primary and secondary key, respectively. 	$O(m \log m)$
4	<ul style="list-style-type: none"> • Scan the groups of the sorted list. A group is defined as a set of entries $(\langle v_i, v_h \rangle, \theta)$ with equal v_i. • Within each group, combine each pair of consecutive entries: $(\langle v_i, v_h \rangle, \theta_1)$ and $(\langle v_i, v_k \rangle, \theta_2)$, and build a wedge (v_k, v_i, v_h). • Within each group, combine also the last and the first entry. 	$O(m)$
5	<ul style="list-style-type: none"> • Sort the wedge list using v_i and v_h as the primary and secondary key, respectively. 	$O(m \log m)$
6	<ul style="list-style-type: none"> • Mark all wedges as unused. 	$O(m)$
7	<ul style="list-style-type: none"> • Find the next unused wedge $W_1 = (v_1, v_2, v_3)$. • Record the initial region list (W_1), and initialize i to 1. • Mark W_1 as used. • If W_1 can not be found, the algorithm terminates and all regions will be extracted. Go to 10. 	$O(m)$
8	<ul style="list-style-type: none"> • Search for the wedge $W_{i+1} = (v_{i+1}, v_{i+2}, v_{i+3})$ following $W_i = (v_i, v_{i+1}, v_{i+2})$ by means of a binary search in the sorted wedge list using v_{i+1} and v_{i+2} as the primary and secondary key, respectively. • Append W_{i+1} to the region list. 	$O(m \log m)$
9	<ul style="list-style-type: none"> • If W_{i+1} and W_i are contiguous, then the region has been extracted. Go to Step 7. • Else increase i by 1 and Go to Step 8. 	
10.	<ul style="list-style-type: none"> • Delete the pseudo region (the contour). 	

Since Jiang and Bunke have proved: "*Sorting is linear-time transformable to the region extraction problem. Therefore, finding all regions of a plane graph with straight line segments requires $\Omega(m \log m)$ time under the algebraic decision-tree model.*" in their paper, the total complexity of their algorithm is $O(m \log m)$, where m is the number of edges of the given plane graph.

4.3.2.2 Polygon-Division-Based Algorithm

- Input: A list of undirected edges in an arbitrary order.
- Output: A list of junctions for each extracted surface.

Step	Polygon-Division-Based Algorithm	Com.
1	<ul style="list-style-type: none"> • Construct the outer contour P using the Contour Extraction Algorithm. • Mark all edges of the P as traced. • Append P to the polygon list. 	$O(m)$
2	<ul style="list-style-type: none"> • Find a vertex v_s from all P' with more than one untraced edge in the polygon list. • Set $v_1 = v_s$. • Obtain all P's sharing v_1 and form a list pv_1. 	$O(m)$
3	<ul style="list-style-type: none"> • Get one of the untraced edge $\langle v_s, v_e \rangle$ starting from v_s. • Mark the edge as traced. • Obtain all P's sharing v_e and form a list pv_e. 	$O(c)$
4	<ul style="list-style-type: none"> • Find the intersection of pv_1 and pv_e. 	$O(c)$

case, size of CEL (Common Edge List) is approximating to the number of total edges of the given line-drawing and thus the most of the computation cost is spent on constructing CEL. On the other hand, size of CEL is very small, i.e. 1, contour-finding takes the largest amount of computation time.

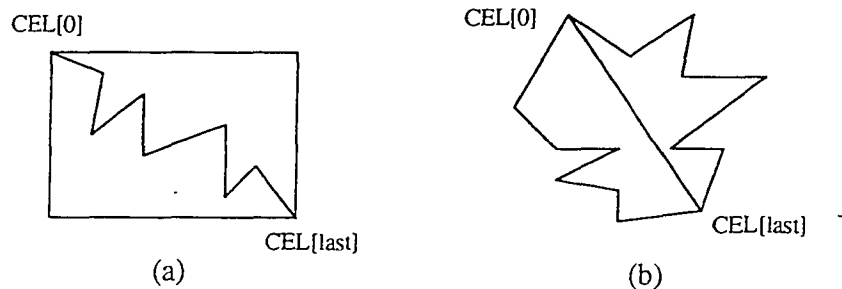


Figure 4.4 - Two Extreme Cases (a) Size of CEL \approx #E (b) Size of CEL \approx 0

4.3.2.3 Comparison

The following is an example of the two different solutions step-by-step. From the example, the efficiency and simplicity (in terms of implementation) of this new algorithm could be obviously seen and measured.

(A) **Jiang and Bunke's Algorithm:** (Six steps are required.)

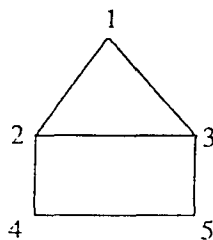


Figure 4.5 - An Example

(1) **Angular Computation (v_1, v_2, θ)**

(1,2,225°), (2,1,45°), (1,3,315°), (3,1,135°), (2,3,0°), (3,2,180°), (2,4,270°), (4,2,90°),
(3,5,270°), (5,3,90°), (4,5,0°), (5,4,180°).

(2) **Sorted by v_1 and θ**

(1,2,225°), (1,3,315°), (2,3,0°), (2,1,45°), (2,4,270°), (3,1,135°), (3,2,180°),
(3,5,270°), (4,5,0°), (4,2,90°), (5,3,90°), (5,4,180°).

(3) **Extracted Wedges (v_1, v_2, v_3)**

(3,1,2), (2,1,3), (1,2,3), (4,2,1), (3,2,4), (2,3,1), (5,3,2), (1,3,5), (2,4,5), (5,4,2),
(4,5,3), (3,5,4).

(4) **Sorted by v_1 and v_2**

(1,2,3), (1,3,5), (2,1,3), (2,3,1), (2,4,5), (3,1,2), (3,2,4), (3,5,4), (4,2,1), (4,5,3),
(5,3,2), (5,4,2).

(5) **Extracted Regions by Searching Loops from the Sorted Wedges**

$R_1=(1,2,3)$, $R_2=(1,3,5,4,2)$, $R_3=(2,4,5,3)$

(6) **Delete the Pseudo Region: R_2 .**

(B) **Polygon-Divisible-Based Algorithm: (Needs two steps only.)**

(1) **Find Outer Contour:** Start from the rightmost and lowest junction, 5, and search in a counterclockwise direction.

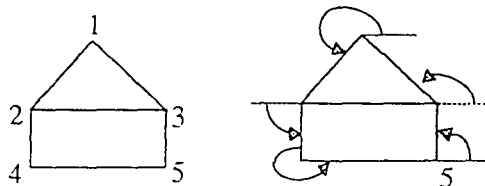


Figure 4.6 - Find Outer Contour

(2) **Start dividing** if a divider can be found.

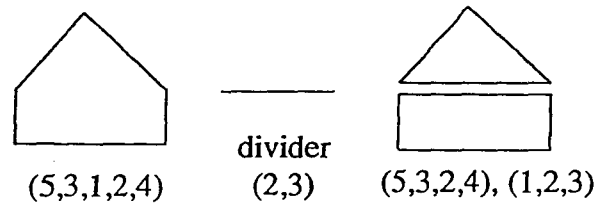


Figure 4.7 - Find_Divider(·) and Do_Divide(·)

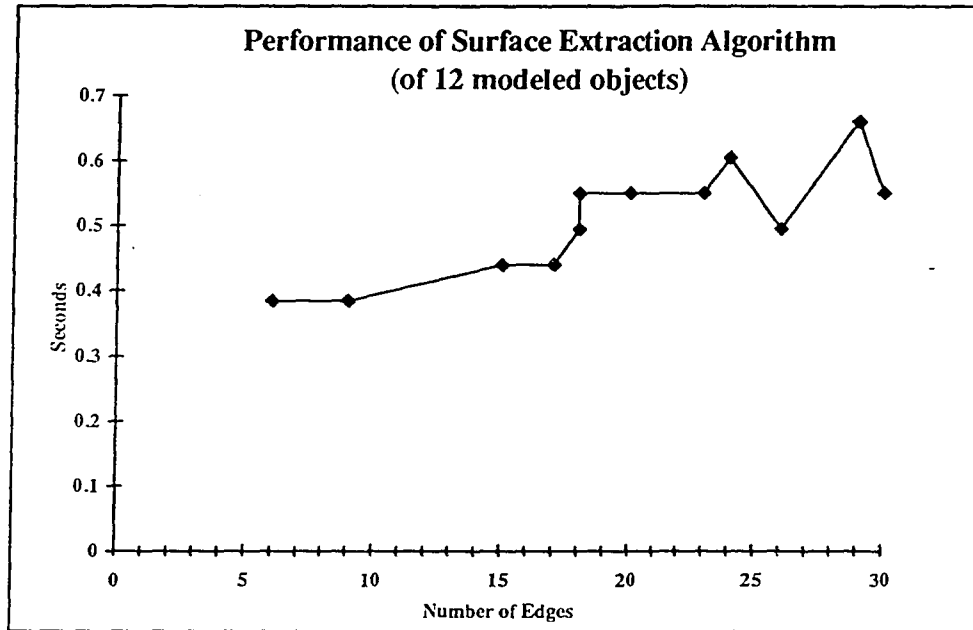
The simplicity and elegance of the polygon-division-based algorithm, in terms of efficiency, has been clearly demonstrated in the above example. In summary, there are many advantages of this algorithm over others (Those have been mentioned in section 4.2.2.) They are listed them below:

- **Easy implementation:** Based on the given data structure, only three main functions are needed. They are `Test_If_Divisible(·)`, `Find_Divider(·)`, and `Do_Divide(·)`.
- No angular computation is required.
- No sorting is required.
- No pseudo surface is generated and thus no extra deletion is needed.
- Automatic hole detection (as explained in section 4.3.1).

4.4 EXPERIMENTAL RESULTS

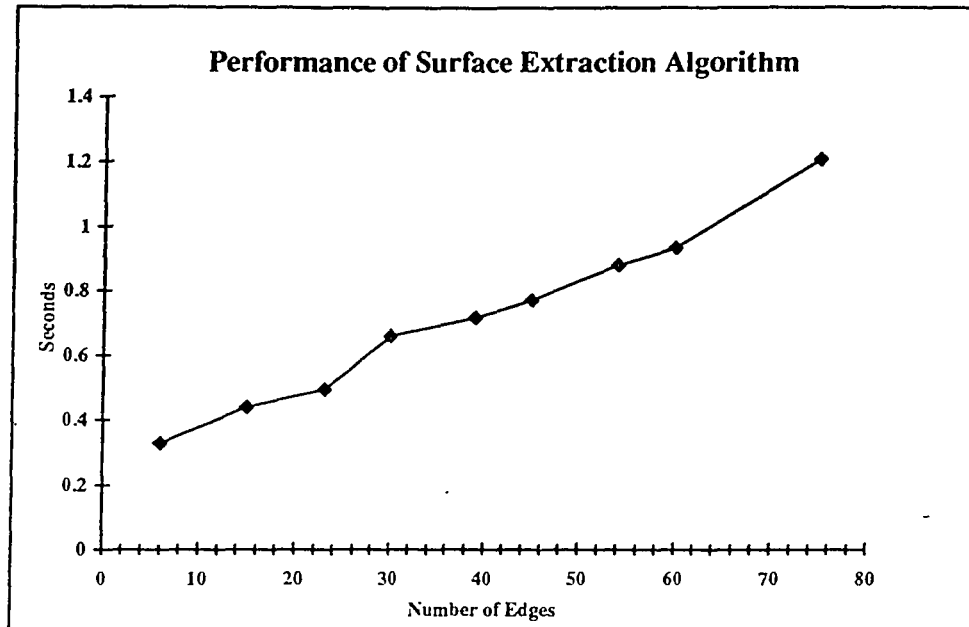
The algorithm has been implemented by C++ language on a 486-PC. The total number of line segments of the test data are ranged from 6 to 75. Figure 4.8(a) shows the execution time of 12 test data (drawings) which are listed in Figure B.1 of Appendix B. The objects of those drawings are the models for the recognition. Since the number of

line segments of these twelve line-drawings are randomly distributed, to show the linear-time performance of this algorithm, a designed set of nine line-drawings are tested and the corresponding performance chart is shown in Figure 4.9(b).



(a)

Figure 4.8 - Performance of the Surface Extraction Algorithm



(b)

Figure 48 - Performance of the Surface Extraction Algorithm (contd.)

4.4 SUMMARY

A practical linear-time algorithm for extracting surfaces from the line-drawing of trihedral objects have been developed. Searching for divisible polygons, finding the divider of the divisible polygons, and then dividing the polygon into two separated smaller polygons are the three major steps of this polygon-division-based algorithm. The algorithm will terminate once no untraced line segments can be found. As the object surface is one of the most important and necessary features for symbolically describing a 3D object, the efficiency of this surface extraction algorithm significantly speeds up the modeling and recognition process.

CHAPTER 5

LINE LABELING AND JUNCTION LABELING

5.1 INTRODUCTION

The intelligent labeling of junctions and lines for a line-drawing has been a long standing challenge in the field of object recognition, computer vision and artificial intelligence. Previous approaches often assume a pre-processing phase that labels the junctions first, followed by line labeling. T. Regier [Regi91] pointed out that junction types cannot be realistically distinguished merely by angle between edges due to limited numerical precision. This chapter will present an approach where both the junction and line labeling processes are executed in a round-robin manner. Each labeling process uses the *Constrained Resource Planning (CRP)* model to pass the results to the other, allowing both processes to work in conjunction until all lines and junctions have been consistently labeled. The computational complexity of CRP has been proven to be linear with respect to the number of tasks (in this case, the total number of lines plus junctions), assuming that few backtracking steps are needed. Experimental results have proven that *Cascaded CRP (CCRP)* not only validated this "near-linear" performance of this approach but also displayed a predominantly high number of test cases that resulted in zero backtracking.

The line-drawing labeling is a classic image understanding problem. However, since it is well known to belong to the class of NP-complete problems, little effort has been directed to the applications of these 2D labeled line-drawing images. In contrast, the near-linear-time computational complexity of this CRP-based labeling approach leads to enable extensive usage of these symbolic labels for 3D object modeling and recognition.

5.2 PROBLEM DEFINITION AND RELATED WORK

5.2.1 Line-Drawing Labeling Problem

The line-drawing of an image is a representation of discontinuities in the intensity of the image. Information pertaining to one view of a 3D object projection is sometimes called a 3D line-drawing image. Recovering 3D information such as depth, surface and object types from a single 3D line-drawing image has always been a very difficult problem. Presently, this is still an open problem owing to the loss of spatial information. Line-drawing labeling allows the understanding of a 3D object in a scene by interpreting line and junction types as surface intersections. Line-drawing labeling is a classical image understanding problem often mentioned in computer vision and artificial intelligence research literatures. It can be formulated into one of the *Constraint Satisfaction Problems* (CSP) that is typically defined as the problem of finding (searching) consistent assignment of values (labels) to a fixed set of variables under some given constraints. Formally, a general labeling (or consistent labeling) problem is defined as follows:

Let $A = \{a_1, \dots, a_n\}$ be a set of units or variables to be labeled, $L = \{l_1, \dots, l_m\}$ a set of labels, T a set of variable constraint relations, and R a set of label compatibility relations. A consistent labeling is one which satisfies all compatibility relations. The labeling problem is to find all consistent labelings of variables (l_{j1}, \dots, l_{jn}) , where $l_{ji} \in L_i$ and $i = 1, \dots, n$ [HarS79].

Since CSP is known to be NP-complete, searching for consistent labels in a given line drawing is also an NP-complete problem. The labeled line-drawing provides or presents basic 3D information of the corresponding objects in a scene from a single

projection. In order to utilize this information for various applications such as 3D object modeling and recognition, a robust and efficient labeling approach should be developed.

5.2.2 Previous Work

The major analytical focus in 3D line-drawing interpretation has previously been on labeling lines and assigning depth attributes to different object domains. Guzman [Guzm68] first developed a SEE program to partition line-drawings into objects by assigning different labels to lines in 1968. However, he did not give a 3D description of the scene. The Huffman-Clowes labeling scheme, which was developed independently by Huffman and Clowes in 1971, [Huff71][Clow71] limited Guzman's work to a trihedral polyhedron, i.e. objects with exactly three plane surfaces coming together at each vertex. They defined a set of possible labels for four different types of junction: L's, T's, Y's and W's (Arrows). Huffman-Clowes' work is so important and famous because it represents the first non-heuristic approach to the line-drawing interpretation. They built up a junction labeling dictionary by enumerating all possible labels in trihedral polyhedron domain. Their work will be introduced in section 5.3.1 in length since our work is also based on their labeling dictionary. Waltz [Walt72] extended Huffman-Clowes labeling dictionary to include shadow boundaries and cracks while adopting a constraint propagation algorithm to label line drawings. The worst-case performance of Waltz's algorithm has been proven to be exponential time [Kiro90]. Mackworth [Mack73] developed a POLY program that can interpret line-drawings and remove unrealizable labelings by reasoning about the gradients (orientations) of surfaces. Based on Huffman-Clowes' junction dictionary, Kanade's approach [Kana80] is able to deal with polygonal planar surfaces such as paper-made objects. He constructed a list of possible junctions for vertices having three or less planar surfaces and demonstrated the validity of the labeling scheme to this new object world, *origami world*. Malik [Mali87] extended Huffman-

Clowes approach to deal with both polyhedral and curved objects. Sugihara [Sugi86] derived a set of algebraic equalities and inequalities, which are both necessary and sufficient for judging the correctness (realizability) of a labeled line drawing under orthographic projection. However, Nalwa argued that Sugihara only provided an exceptionally weak claim [Nalw88]. In the mean time, Nalwa has tried to give a mathematical theory of line-drawing interpretation. He developed a series of theorems to prove that straight lines and conic sections in line-drawings are projections of their respective scene edge counterparts. An excellent generalized mapping from orthographic to perspective projection was also described in his paper. Recently, many neural network approaches were proposed to speed up the conventional line labeling algorithms [LiYD88][TsaL90][SalY91]. Cooper [Coop93] has extended the work of Malik on curved objects with piecewise C^3 surfaces. Lamb and Bandopadhyay [Lamb93] have proposed a heuristic rule to reject many geometrically impossible interpretations.

5.2.3 Motivation and the Proposed Solution

The line-drawing labeling solution via a single CRP model has been developed and demonstrated under the assumption that each junction type has already been correctly determined in the pre-processing stage [Chen91]. It was pointed out by T. Regier that in practical cases the junction type cannot be distinguished merely by the angle size between edges because of the precision errors in numerical computation [Regi91]. A misclassification, e.g. classifying a Y- as a T- or W-junction, will result in a totally different line labeling and even unsolved configurations (Figure 5.1). For example, there is one solution in Figure 5.1(a) and no solution for 5.1(b) and 5.1(c) even though all three neighboring junctions are correctly classified as W-junctions.

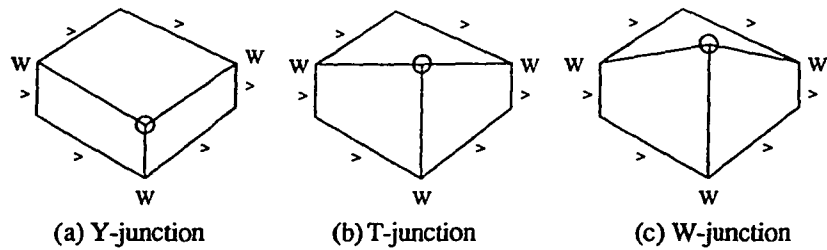


Figure 5.1 - Different Junction Interpretation Results Different Line Labels

Regier has proposed a solution for line and junction labelings by using a coupled probabilistic system which performs line/junction detection and labeling by two Markov Random Fields (MRFs). The labeling MRF computes likelihoods (probabilities) of junction types or line labels from the neighborhood graph and a set of cliques are collected. However, the success of his approach depends highly on the prior probabilities and line roles in different junction types. Besides, only a few simple test results have been shown in his paper.

In this chapter, a Cascaded CRP model that can efficiently solve both junction and line labeling problems in an alternating manner will be introduced [CheY94b]. The Junction Labeling CRP (JCRP) procedure determines the junction types from the labeled connecting lines and the Line Labeling CRP (LCRP) procedure assigns line labels to the connecting lines of the known junction types. Each CRP passes the computed results to the other and this continues until all lines and junctions have been consistently labeled. Since most of the experiments have no backtracking at all, the Cascaded CRP solution approaches to near-linear time performance.

The remaining sections of this chapter is organized as follows. In section 5.3, the Huffman-Clowes labeling scheme and basic operations of the general CRP model will be briefly reviewed. Six fundamental concepts of Cascaded CRP model for both line and junction labeling are given in section 5.4, along with the overall algorithm. A traced

example and two experimental results are shown in section 5.5. The achievements and the usage of the resulting labeled line-drawing will be explained in the concluding section.

5.3 PRELIMINARIES

5.3.1 Review of Huffman-Clowes Labeling Scheme

In the Huffman-Clowes labeling scheme, lines in images are classified into three categories: convex lines (represented by plus labels, '+'), concave lines (represented by minus labels, '-') and occluded lines (represented by arrow labels, '>'). A convex line lies at the intersection of two surfaces perceived as a ridge. On the other hand, a concave line lies at the bottom of a valley formed by two intersecting surfaces. The direction of the arrow label on the boundary lines depends on whether its right-hand side of the line is visible. Since the junctions in 2D line-drawings correspond to vertices in the physical 3D objects, they can be categorized according to the number of lines coming together and the angle size between the lines. In other words, the junctions can be classified according to the number of intersecting surfaces at the vertex. The four types of junction are called L-, Y-, T- and W-junctions and are defined as follows: Angles between every two lines in a Y-junction are smaller than 180° ; if one of the angles equals 180° , it is called a T-junction; if one of the angles is greater than 180° then it is called a W-junction. Formally, the lines for W- and T-junctions are defined in Figure 5.2. They are useful in the following proofs and in chapter 6. In addition, their possible labels defined by Huffman and Clowes are shown in Figure 5.3.

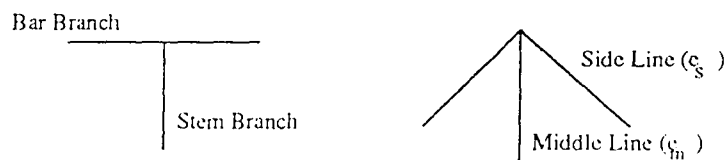


Figure 5.2 - Edge Definitions of T- and W-junctions

The following assumptions in the Huffman-Clowes scheme restrict the number of junction types to four, and the application domain to trihedral polyhedron or trihedron.

1. Curved objects are not considered.
2. There are no shadows or cracks in the drawings.
3. All vertices are intersections of exactly three object surfaces, i.e. three-face vertices.
4. Drawings are captured at a general view point, i.e. no junction will change its type with minute eye movement and orthogonal projection.
5. The object is solid and opaque; no hanging faces or edges are allowed and no hidden line are displayed in the image.

















L-junctions			Y-junctions		W-junctions		
2 faces	2 faces	1 face	3 faces	3 faces	3 faces	3 faces	2 faces
							
(A)	(B)	(C)	(G)	(H)	(N)	(O)	(P)
			T-junctions				
2 faces	2 faces	1 face	2 faces	3 faces	2 faces	2 faces	3 faces
							
(D)	(E)	(F)	(I)	(J)	(K)	(L)	(M)

Figure 5.3 - Huffman-Clowes' Labeling Dictionary

Figure 5.4 shows the explanations of the different types of junction labels in some actual line-drawings. As one can observe from the definitions of junction types, it is not possible to make such a clear distinction between angles greater than, equal to or less than 180° due to numerical errors in angular computation or noise in the original images. This can result in coordinate-shifts during the edge-extraction process and therefore junction

types could not be determined directly. As a result, line labeling has been found to be helpful in solving the junction labeling problem as well. This observation leads to the approach of cascading the two labeling processes in an alternating manner.

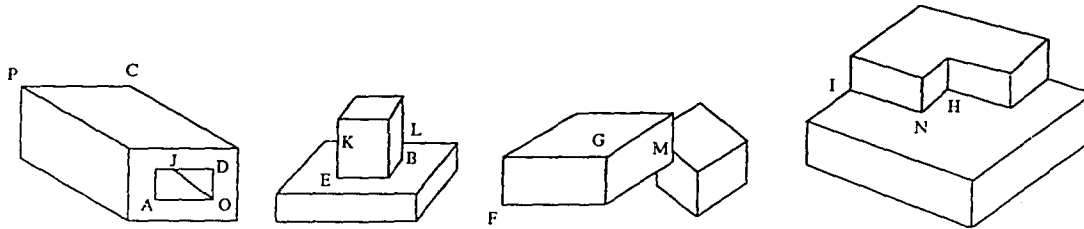


Figure 5.4 - The Corresponding Junctions in Realistic Line-Drawings

In essence, this research work is also based on Huffman-Clowes' junction labeling dictionary but a novel labeling algorithm, *constrained resource planning* [KenY89] is used. The whole line drawing is treated as the resource and the task is a label assignment. Solution from the Huffman-Clowes labeling dictionary with consistency constraints could then be chosen for each unassigned task. However, as mentioned before, those solutions are developed on the basis of pre-classified junction types. Once a junction type is determined, the line labeling process can proceed. On the other hand, a junction type can be determined when its line labels are already assigned. In other words, the two processes rely on one another.

5.3.2 Constrained Resource Planning (CRP) Model

The CRP model, based on the original work of Keng and Yun in 1989 [KenY89], has already been firmly established as a broadly applicable technique to solve numerous resource management problems. The model is a general and effective methodology to solve resource management problems by two domain-independent guiding principles, called *most-constrained* and *least-impact* strategies. At each iteration, the CRP model

shown in Figure 5.5 can be simply described by a four-corner loop that includes 1) choosing the most constrained task from an agenda; 2) constructing the solution space; 3) selecting the least impact solution; and 4) generating a new agenda by constraint propagation. Resources come from the collection of usable elemental units to perform the tasks of a given problem. The set of active tasks is called a task agenda. A solution to a task is a collection of resource units assigned to perform a selected task. The solution space for a task comprises all possible solutions of that task.

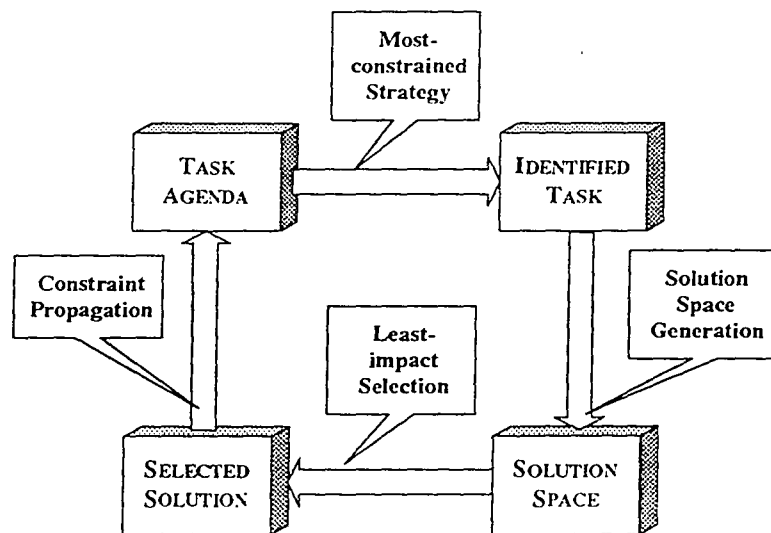


Figure 5.5 - Configuration of the CRP Model

The *most-constrained strategy* guides the task identification process by choosing the task with the least number of possible solutions, or most critical task which has the least flexibility for delay and hence has the highest priority, i.e. criticality value. The *least-impact strategy* selects the solution of the current task by minimizing the impact to other tasks, as a result, maximizing the feasibility of completing all remaining tasks. The impact of a solution on others is measured by the cruciality function, which estimates the total demand on the resource units it utilizes. The algorithm is shown as follows [Keng89].

-
1. Construct the task agenda $T = [T_i]$;
 2. While T is not empty { /* Four corner loop starts */
 3. Compute the criticality Ω_i for task T_i ;
 4. $\Omega_i = 1/D_i$, where D_i is the number of solutions of task T_i ;
 5. Select a task T_i with highest criticality; /* Select the most critical task */
 6. Compute the resource demand γ_k for resource unit r_k ;
 7. $\gamma_k = \sum_i \delta_i$, where $\delta_i = \begin{cases} 1, & \text{if } r_k \in D_i \\ 0, & \text{otherwise} \end{cases}$ /* Solution space formulation */
 8. For the selected task T_i , compute the cruciality Γ_j for each solution P_{ij}
 9. $\Gamma_{ij} = \sum_k \omega_k \gamma_k$, where $\omega_k = \begin{cases} 1, & \text{if } r_k \in P_{ij} \\ 0, & \text{otherwise} \end{cases}$
 10. Select a solution P_{ij} with the lowest cruciality; /* Select the least impact solution */
 11. Constraint propagation; } /* Constraint Propagation */
-

The CRP model has shown its prowess in solving many resource management problems such as the traveling salesman problem [YuQC92], the switchbox routing problem [HoYH85], and job-shop scheduling [KeYR88].

5.4 LINE AND JUNCTION LABELINGS BY CASCADED CRP (CCRP) MODEL

5.4.1 Architecture of CCRP Model

From previous experience, it is not easy to construct two domain-independent heuristics, i.e. selecting the most-constrained task and the least-impact solution of CRP for solving a complex problem. If there are two (or more than two) sets of tasks which consume the same resources but are constrained to each other, a CRP solution can be

developed for each of them independently and exchange their constraints. This is the basic motivation in developing such a Cascaded CRP solution for this kind of problem. This concept is then examined and implemented by using a cascaded CRP architecture, shown in Figure 5.5, for both line and junction labelings simultaneously.

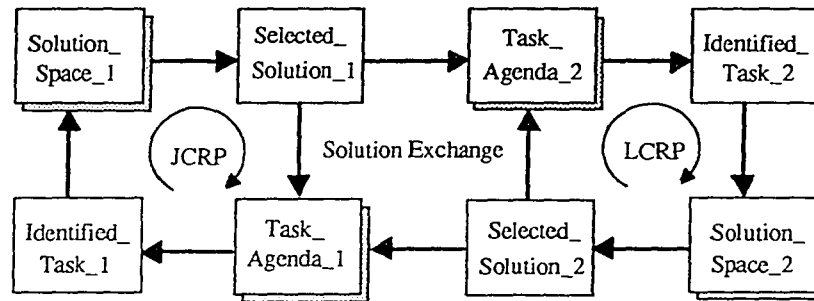


Figure 5.6 - Architecture of Cascaded CRP Model

The system will start after some *pre-processing* operations on the lines and junctions. For lines, boundary lines (contour of the junction graph) will be labeled first because they have only one solution, ">". A different interpretation can be made when all boundary lines are labeled with "-". In this case, the base of the object is connected to a infinite-large background plane. This interpretation is excluded from normal cases. An object is always located at the right-hand side of the boundary if one traces the outer contour of the junction graph in a clockwise direction. For junctions, L-junctions (two-line junctions) can be labeled directly.

The outputs (constraints) of these two CRPs are propagated to both Agenda_1 and Agenda_2 immediately because they are useful in the selection of the next task. The following table shows six essential concepts used in both line and junction labeling CRPs. The two CRPs, LCRP and JCRP are both constrained by line-drawing's geometry consume the same resources, Huffman-Clowes' labeling dictionary, but there is no resource competition between them because they are executed interactively. Their

constraints and omain-independent measurement functions - criticality and cruciality, are also very similar. However, their tasks and solutions are different.

Table 5.1 - Six Concepts for Both CRPs

	Line Labeling CRP	Junction Labeling CRP
Resource	Possible line label set	Possible junction label set
Task	Unlabeled lines of a specific junction	Unlabeled junctions
Solution	A consistent line label assignment based on the known junction label	A correct junction label assignment to the junction with consistent line labels.
Constraint	Labels for the two end junctions	Line labels incident to the junction
Criticality	Number of possible line labels	Number of possible junction labels
Cruciality	$1/\sum$ (number of possible line labels for unlabeled lines of the neighboring junctions to a specific junction)	$1/\sum$ (number of possible junction labels for unlabeled neighboring junctions to a specific line label)

5.4.2 Prior Knowledge for Selecting Solutions

The following prior knowledge help reduced the number of solutions as well as speeds up labeling processes during the cruciality calculation.

- (1) The middle line (e_m) of W-junctions can only be labeled by either '+' or '-'.
- (2) There is at least one line being labeled by '+' or '-' for Y- and W-junctions.
- (3) Whenever a Y- or T-junction has the same line labels such as (I) and (J) in Figure 5.3, it will be classified as a Y-junction given that it is shared by only two visible faces in the given line-drawing. Otherwise, it will be classified as a T-junction.
- (4) Similarly, when a W- or T-junction has the same line labels such as (E) and (K) in Figure 5.3, it will be classified as a W-junction provided that it is shared by two

visible faces in the given line-drawing. Otherwise, it will be classified as a T-junction.

- (5) There is no ambiguity between W- and Y-junctions because they are separated far away, i.e. transitions are always from Y- to T- or T- to W-.
- (6) In the projection of right-angle trihedron, all neighboring junctions of a Y-junction are three-line junctions, but not vice versa.

Lemma 5.1 shows if all neighbors of a three-line junction (T-, Y-, or W-) are W-, the three-line junction will be a Y-junction. Lemma 5.2 further concludes that a Y-junctions will never have a L- neighbor in the right-angle trihedrons. In other words, in the orthographic projection of right-angle trihedral objects, all neighboring junctions of a Y-junction are three-line junctions. This is also a proof of the prior knowledge (6).

Lemma 5.1: In a single-object scene, if all neighbors of a three-line junction are W-junctions, the three-line junction will be a Y-junction. Conversely, under the same assumptions, if all three neighbors of a Y-junction are three-line junctions, they are not necessarily all W-junctions.

Proof:

It is noted that the middle line and the two side lines of W-junction are defined as e_m and e_s respectively, as shown in Figure 5.2.

(1) It is always true for " \Rightarrow ":

In Figure 5.7, +, -, and > represent the labels for convex, concave and occluding edges respectively. The number of the middle lines (e_m) and side lines (e_s) of neighboring W-junctions connected to the given three-line junction is drawn at the bottom of each sample drawing in the following figures.

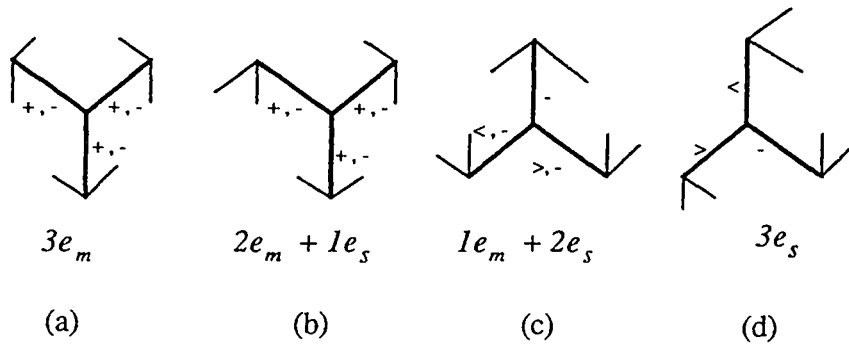


Figure 5.7 - Case 1: One Y- and Three W-junctions

In the first case, shown in Figure 5.7, if the three-line junction is a Y-junction, there are only four different combinations of the three W- neighboring junctions.

If the three-line junction is a T-junction then there are only two exactly types of connections, as shown in Figure 5.8. This is because the middle line of a W-junction can never have a ">" or "<" label but the two bar-branches of a T-junction can only be labeled by ">" and "<". In addition, these situations can only happen in multi-object scenes.

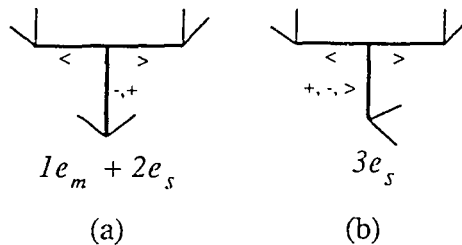


Figure 5.8 - Case 2: One T- and Three W-junctions

In the third case, if the three-line junction is a W-junction, its connections with all W- neighbors are shown below (Figure 5.9), where ①, ②, and ③ represent three faces that intersect at the three-line junction.

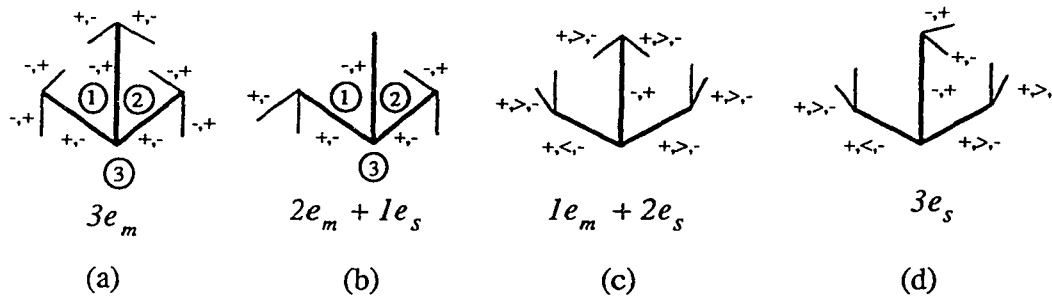


Figure 5.9 - Case 3: Four W-junctions

In the first of two drawings in Figure 5.9, (a) and (b), the third face should be broken into two faces. The drawing in Figure 5.9(c) has valid labels but it can only happen in an extreme view (we will define an extreme view in chapter 6) under a perspective projection, as shown below.

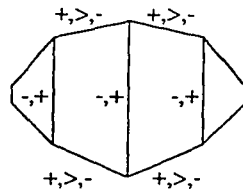


Figure 5.10 - Extreme View of a Perspective Projection

Similarly, the fourth drawing (Figure 5.9(d)) does not exist due to convexness/concaveness. Therefore, it can be concluded that *if all neighbors of a three-line junction are of W-, the three-line junction must be a Y-junction in the non-extreme view of a single-object scene.*

(2) Conversely, " \Leftarrow " is not true and can be exemplified by the followings contradictory examples for multi-objects and single-object scenes.

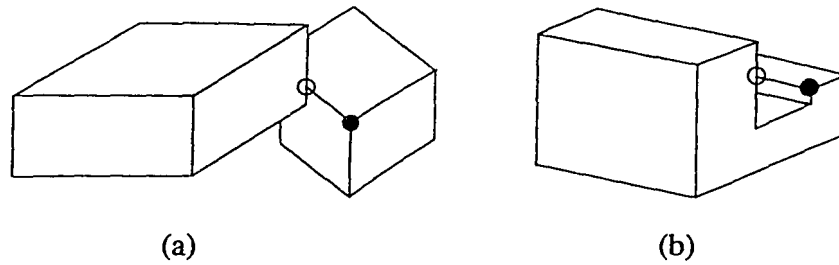


Figure 5.11 - Contradictory Examples

In the line-drawings of Figure 5.11, one of the neighbors of the Y-junction, indicated by the filled circle, is a T-junction. Therefore, " \leq " is not true. However, the following conclusion can still be obtained: *all neighboring junctions of a Y-junction are three-line junctions (i.e. Y-, T-, or W-) in the non-extreme orthographic projection of right-angle trihedral objects.* **Q.E.D.**

Lemma 5.2: *Y-junction will never have an L- neighbor in any projection of right-angle trihedrons. ■*

Proof:

(1) If a Y-junction has either all "+" labels or all "-" labels, then all possible connections of Y-L pairs are shown in Figure 5.12. For the L-junction, face ① is a background face. However, for the Y-junction, it is not. Therefore, these connections will never exist in real scenes.

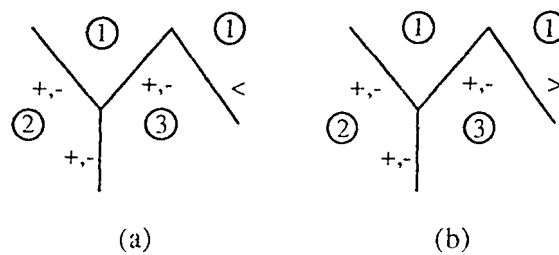


Figure 5.12 - A Y-L Pair for a Convex or Concave Y-junction

(2) If the Y-junction has two arrow labels and one minus label, then the possible connections to the L-junctions are listed in Figure 5.13. In the two leftmost figures, using the same reasoning as above, face ① is the background of the L-junction but not for the Y-junction. In the two rightmost figures, face ③ is a background face of Y-junction but face ① is also a background face of the L-junction. So, they contradict each other.

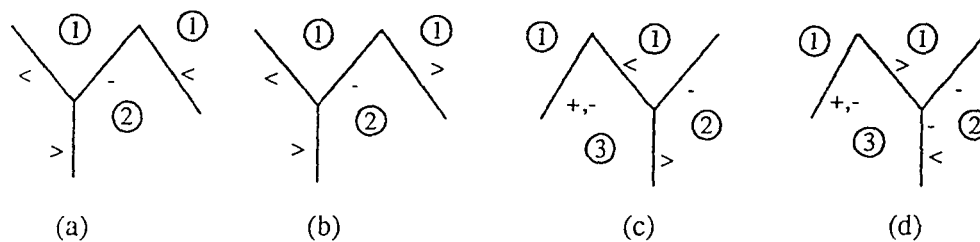


Figure 5.13 - Y-L Pair for a Y-junction with Different Labeling

A contradictory example can be found for **any trihedrons** (not limited to right-angle trihedrons), as shown in Figure 5.14.

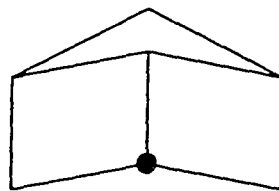


Figure 5.14 - An Contradictory Example for Any Trihedrons

Q.E.D.

5.4.3 Planning-based Labeling Algorithm

5.4.3.1 Main Function

The input to our labeling system is a line-drawing represented by polygon, edge, and junction tables. Given a drawing, the outer contour of this drawing is extracted first

and assign '>' labels to each line belonging to it. The contour extraction algorithm has been presented in section 4.3.1.1 (chapter 4) of this dissertation. L-junctions can be labeled directly because there are only two connecting lines. Some of the junction labels which have passed the pre-classification rules can be classified or identified before entering CCRP loop. The pre-classification rules will be described in section 5.4.3.2. The first single line labeling CRP is applied to junctions of known types. The remaining unlabeled junctions and lines are then fed into the main part of our system, CCRP loop, where junction types and line labels are determined alternatingly between JCRP and LCRP until all tasks (junctions and lines) are completed or have consistent labels. The overall flow-chart of the algorithm is shown in Figure 5.15.

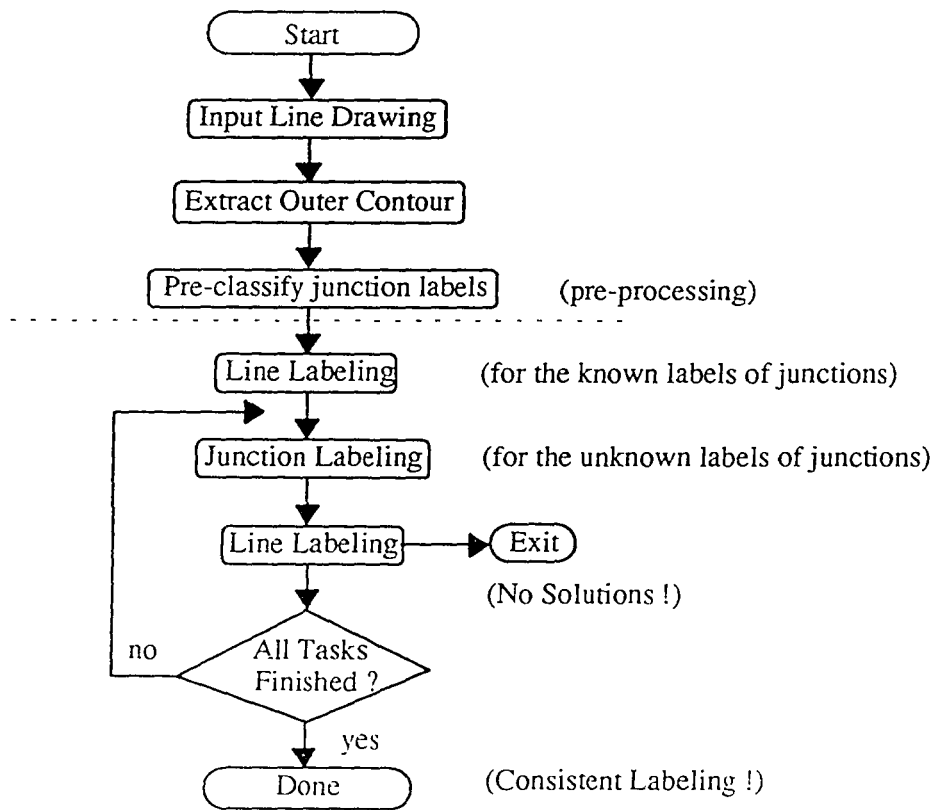


Figure 5.15 - Flow-chart of Cascaded Junction/Line Labeling CRP

5.4.3.2 Pre-Classification Rules of Junction Labels

To provide enough clues in accomplishing the whole labeling process, some of the junction types will be classified solely based on the angle sizes rather than relying on their line labels. From the definition of junction types, a junction type is determined according to the largest angle between two junction edges. The range of angle sizes is then divided into 5 regions within $[0, 2\pi]$: Y-, TY-, T-, WT-, and W-. It is noted that WT- and YT- are two fuzzy regions for classification purposes. The junctions that fall into these regions need to be classified by the following Cascaded CRP.

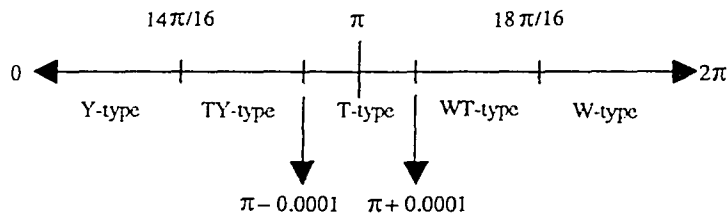


Figure 5.16 - Thresholds for Junction Type Pre-Classification

5.4.4 Data Representation

The first three tables: polygon, junction and edge tables, are commonly used in both CRPs and are implemented by a linked-list structure. The polygon table contains all 2D polygons which are projections of 3D surfaces. Edge and junction tables store line and vertices information such as junction labels, edge lists and coordinates. The only difference between JCRP and LCRP is in their solution types. In JCRP, the solution comprises junction indexes and the corresponding type. In LCRP, the solution includes all line indices and the corresponding labels for a specific junction.

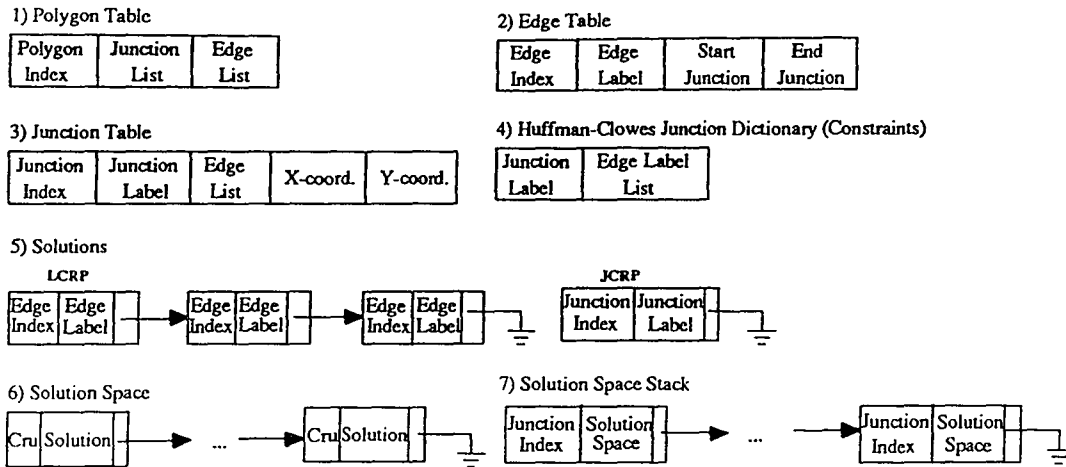


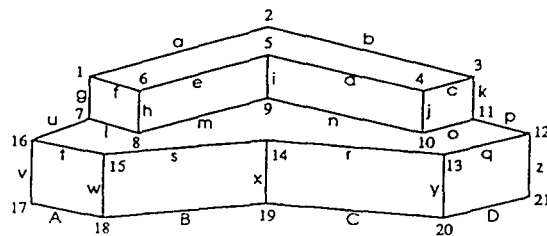
Figure 5.17 - The Data Structure Used for Junction/Line Labeling Cascaded CRP

5.5 EXPERIMENTAL RESULTS

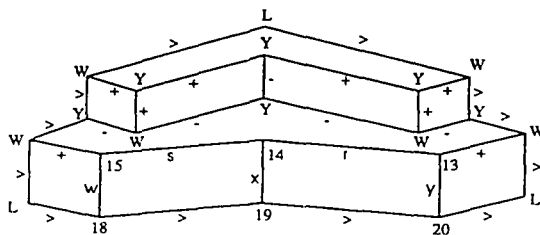
5.5.1 An Example

A simple example is given in this section and the corresponding input-drawing is shown in Figure 5.18(a). After pre-processing, lines a, b, k, p, z, D, C, B, A, v, u, and g have arrow labels because they all belong to the outer contour of the given drawing. Consequently, junctions 2, 21 and 17 are classified as L-junctions. Since junctions 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16 are apparently classified into appropriate types in the junction labeling pre-classifying step, the following line labeling CRP (LCRP) can obtain some labeling results from those already known junction types. The resulting (preprocessed) labeled line drawing is shown in Figure 5.18(b). The remaining unlabeled junctions are: 13, 14, 15, 18, 19 and 20. And the unlabeled lines are: r, s, w, x and y. Hence, the agenda of the CCRP is the set of unlabeled junctions (13, 14, 15, 18, 19, 20). The first iteration in the junction CRP, JCRP, will select junction 13 as the most constrained task because the number of solutions for this junction is the smallest among all candidates. Two types of solutions, Y- or T-, are formed after solution formulation. For the solution

of a Y-junction, the corresponding cruciality is 0.33 and the solution of a T-junction has cruciality 0.2. Therefore, W- is selected as the solution for junction 13. After junction 13 is classified into a Y-junction, there is only one line labeling solution for it. In the following LCRP, labels of line r and y can be determined immediately. Prior knowledge (4) is useful for junction label classification. According rules (4) and (5), junction 18, 19, and 20 have labels W, Y and W respectively.



(a)



(b)

Figure 5.18 - Example of (a) Original Drawing (b) Drawing After Pre-processing

5.5.2 Results and Performance Analysis

The following figures show the resulting labeled line drawings by this cascaded CRP approach. The first example has no backtracking (Figure 5.19). The types of six junctions at the center of the lower brick corresponding to junction 13, 14, 15, 18, 19, and 20 in Figure 5.18(a), were determined by the CCRP loop. In the second example, Figure

5.20, one backtracking appears for the T-junction on the upper-right corner of the drawing.

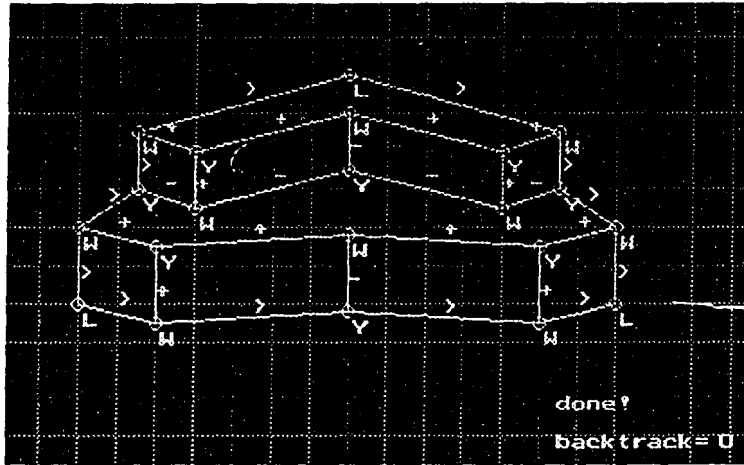


Figure 5.19 - Labeled Line-drawing With No Backtracking

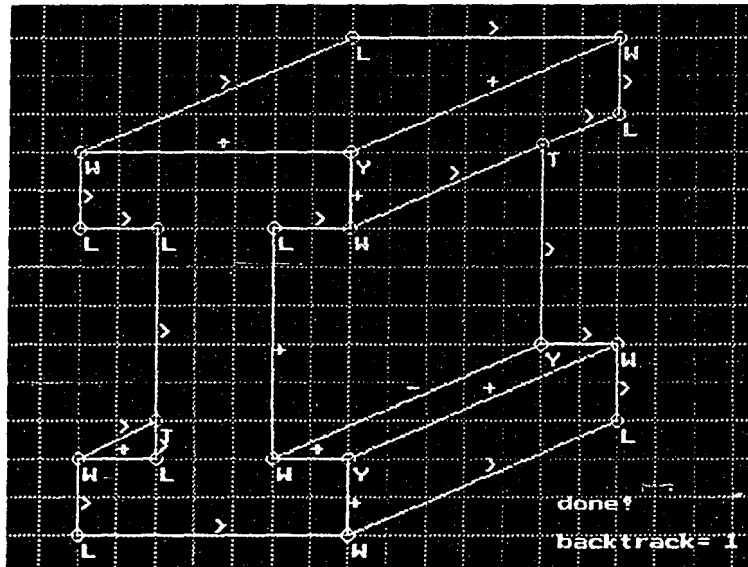


Figure 5.20 - Labeled Line-Drawing with One Backtracking

The above experiments were programmed in C++ language on a 486-PC. The drawing shown in Figure 5.20 takes 1.75 seconds for pre-processing and 0.05 seconds for CCRP. The second drawing, Figure 5.20, spends most of the processing time on the

single CRP because there is only one unknown junction type after pre-processing. Many other test cases have been simulated and shown the robustness and effectiveness of this cascaded planning model in solving the drawing labeling problem successfully. The efficiency of this approach is demonstrated by the small number of backtracking (none in most cases) in all tested cases, thereby supporting the complexity estimate of near-linearity with respect to the number of junctions and lines. The first experiment has been done on 12 line-drawings of modeled objects. The performance of these 12 test line-drawings is shown in Figure 5.21. Since the numbers of edges plus junctions of these drawings are not distributed evenly, the linear-time complexity of this algorithm is not obviously seen from this figure. These 12 drawings are listed in Figure B-1 of Appendix B. The number of backtracking ($\#B$) of each drawing is also listed in Figure B-1.

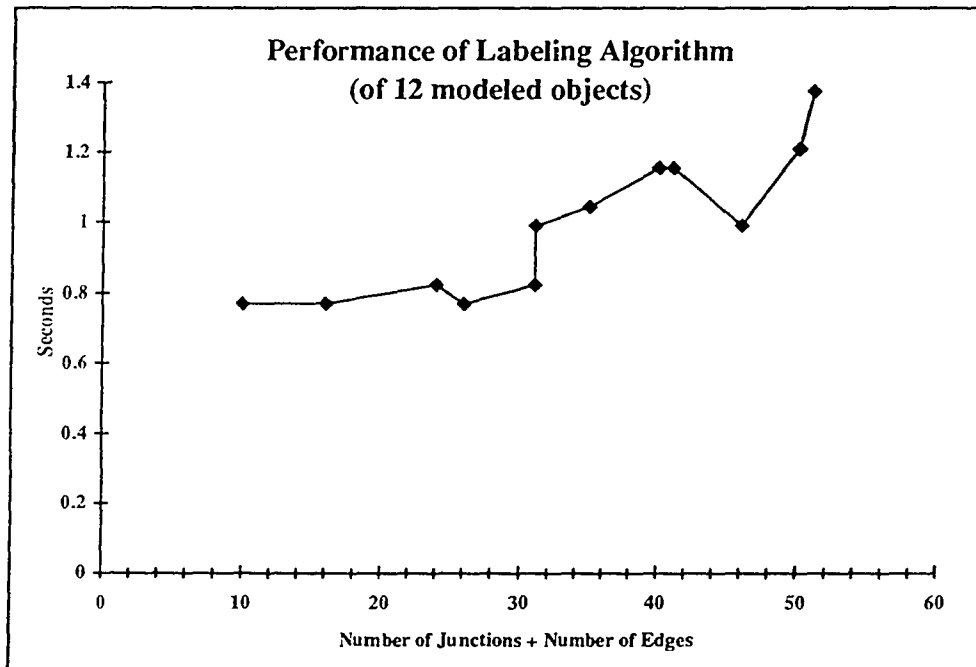


Figure 5.21 - Performance of Labeling Algorithm for 12 Modeled Line-drawings

Figure 5.22 shows the performance of another experiment with a set of designed line-drawings where the numbers of edges plus junctions are ranged from 10 to 126 as uniformly as possible. This experiment can clearly prove a linear-time performance (with respect to the number of lines and junctions) of the CRP-based labeling algorithm.

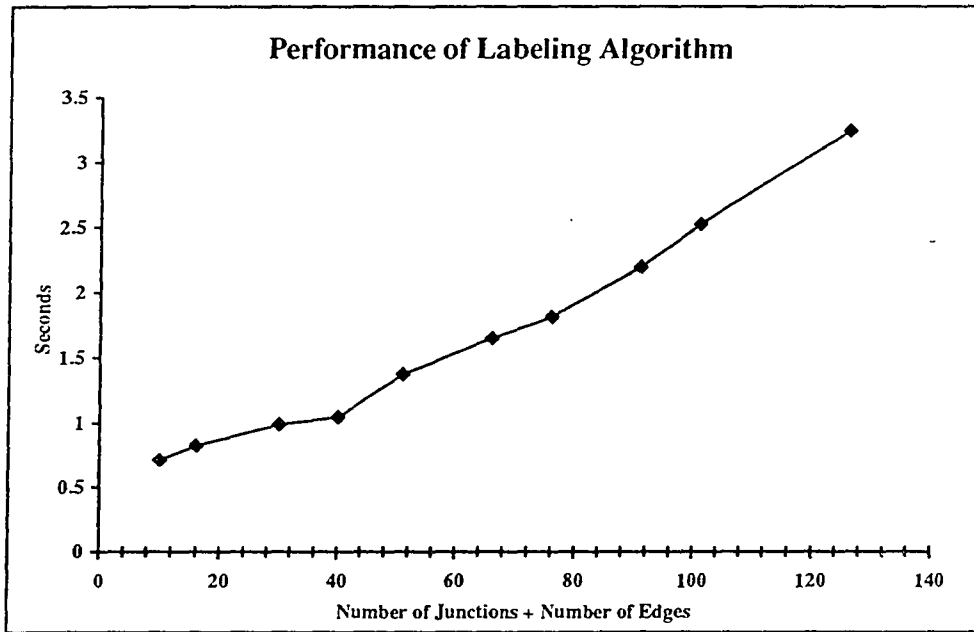


Figure 5.22 - Performance of Labeling Algorithm for Different Problem Sizes

5.6 SUMMARY

Junction and line labels are two of the most important types of information embedded in a given line drawing. However, due to precision errors of numerical computation and noisy data, one will not be able to guarantee correctly or consistently derive labels of the given line-drawings obtained after edge and corner detection and line following on the original image. A novel Cascaded CRP-based labeling approach has been proposed in this paper to efficiently assign consistent labels for both junctions and lines. It

is shown that junction and line labeling works alternately and cooperatively by propagating constraints (solutions) to each other until a consistent labeling is achieved for the whole drawing. The computational complexity of CRP has been determined to be linear with respect to the number of tasks (here, the number of lines plus the number of junctions), provided no backtracking steps are needed. The experimental results of this new Cascaded CRP approach have shown to validate this near-linear performance due to a predominantly high number of test cases resulting in no backtracking at all. Since the junction and line labels are invariant to 2D transformation (translation, scaling and rotation), the resulting (labeled) line-drawing has been shown to be useful and important for both 3D object modeling and matching process in chapter 6.

CHAPTER 6

3D OBJECT MODELING AND RECOGNITION

6.1 INTRODUCTION

This chapter presents a valid-view modeling and multi-view indexing approach for solving the 3D object recognition problem. Based on labels of 2D projections (views) from 3D objects, the way in which this approach achieves less storage space and faster recognition is shown in this chapter. In addition, the multi-view indexing strategy has been demonstrated to provide a more natural and reliable perception process than the traditional single-view approach.

Most of the 3D object recognition systems are model-based, where features extracted from 2D drawings are matched with stored 3D object models. Presently, *viewer-centered*, or *multi-view object representation* is among one of the most popular 3D object modeling approaches. Rosenfeld has made the following conjecture in 1987 [Rose87]:

CONJECTURE 1 - *"For purposes of rapid recognition, humans represent a 3D object by a set of characteristic views or aspects, i.e. by a set of commonly occurring 2D projections."*

Viewer-centered object representation of 3D objects was developed to construct the object models with all possible views based on the above conjecture. However, most of the previous approaches [KorD87][WanF90] define an aspect as a view where it is topologically different from its neighboring views. They do not refer to the "commonly occurring" concept from the above conjecture. Therefore, these approaches often suffer

from three crucial drawbacks, preventing widespread practical usage: 1) large amount of storage space is required; 2) the matching process is extremely time-consuming; and 3) a unique object match is not guaranteed.

In the conventional viewer-centered approach, a 3D object is represented by many (often a large number such as 71 for a simple L-shaped block in [WanF90]) 2D views. Each view is usually represented by either a contour or face graph. Any view of the object can then be matched directly with the views stored in the model by any graph-matching method. As a result, this method of model storage and matching frequently suffers from the excessive storage and computational requirements. Due to its high computing and searching cost, the traditional way of looking for an unknown input object from a collection of modeled 3D objects is to match only one of the 2D input views with the stored 2D views in the model base. However, from a recognition point of view, the following new conjecture may be created for 3D object recognition [CheY94].

CONJECTURE 2 -*"For the purpose of reliable recognition, humans perceive a 3D object usually by looking for a set of different but informative views from different viewpoints."*

To achieve reliable and fast recognition, reducing the size of object models and speeding up the matching process are two primary approaches that can bring a new multi-view 3D object recognition strategy into a reality. A fact is found in this research that not all the views are equally significant. Since it is observed that many views are less informative and often redundant in 3D object recognition, a method to eliminate them from the model base is proposed. All remaining views are labeled and represented by a set of 8-tuple feature vectors, which include the number of visible faces and the numbers of different types of edge labels and junction labels. Each feature vector is converted into a

computable and uniquely identifiable *signature* of the corresponding view, and use it as a much more simplified representation. Searching for an object or one of its views in the model base, then, becomes merely a textual matching process (rather than geometric or graphic matching). As a result, the computation cost for a single view is reduced tremendously.

It is noted that the individual model view of an object may not be unique, however, an object can be uniquely determined by a set of model views, which is called a *canonical view list*. It is obvious that two different objects may have the same views, especially for those occurring in a large model base, but, there must also exist a different view since they are distinct from one another. For example, two different objects in Figure 6.1(a) and 6.1(b) have the same views from a number of viewpoints. However, Figure 6.1(a) will never have a view as shown in 6.1(b).

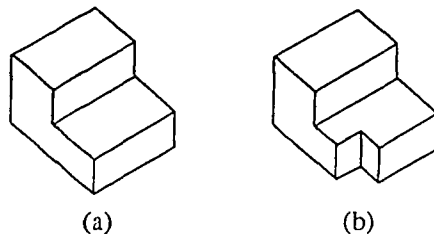


Figure 6.1 - Two Similar Objects

The rest of this chapter is organized as follows. The problem definitions and related work will be reviewed in section 6.2. In section 6.3, some terms of for the new object representation will be defined. A detailed approach of constructing 3D object model is given in section 6.4. A canonical representation of a 3D object is proposed to aid in constructing a minimum set of representative as well as unique views. Based on this new object modeling technique, in section 6.5, a multi-view textual matching approach using the hashing technique is presented. A new weighted confidence computation model is presented. It combines the significance of individual views and the sequences of input

views for the hypothesis generation in the multi-view recognition approach. Lastly, the major achievements and future directions of this research will be discussed.

6.2 PROBLEM DEFINITION AND RELATED WORK

Object-centered and viewer-centered are two major approaches for 3D model-based object representation [BesJ85][ChiD86][BowD90]. In object-centered (or CAD-based) representation, the model is constructed in 3D space by using CAD tools [BhaH87][Fly91]. The input view of a 3D model is first found via a corresponding 3D to 2D transformation which searched for the most similar projection to the input view. Usually, object-centered approach uses less storage space than viewer-centered approach but requires a much higher computation cost for recognition.

In contrast to the object-centered approach, the viewer-centered (or multiview) approach stores all the features of necessary projections. The features of the input view are then matched with all of the projections in the model to find the best match. This approach only requires a simpler matching process but a larger space is needed to store the projections. In other words, as the computation cost shifts from recognition to model construction, it is anticipated that real-time object recognition applications will become more and more feasible in the future. Hence, presently the viewer-centered object modeling approach is more popular and attractive .

Conventional methods of constructing viewer-centered object models can be categorized into two main streams: uniform tessellation and aspect graph. The uniform tessellation approach [KorD87] partitions the viewing space into small and regular regions, such as the one in Figure 6.2. Each region represents a view of the 3D object. Region growing algorithm groups adjacent views if they are topologically equivalent, into bigger regions. The size of the initially partitioned region is very important in this

approach. A large partition may miss some important views; however, on the other hand, a too small partition may need too much computation. The advantage of this approach is that the partition algorithm is simple and is also applicable to all kinds of objects. In the second approach, viewing space is partitioned by sampling the continuous view sphere surrounding an object at discrete viewpoints.

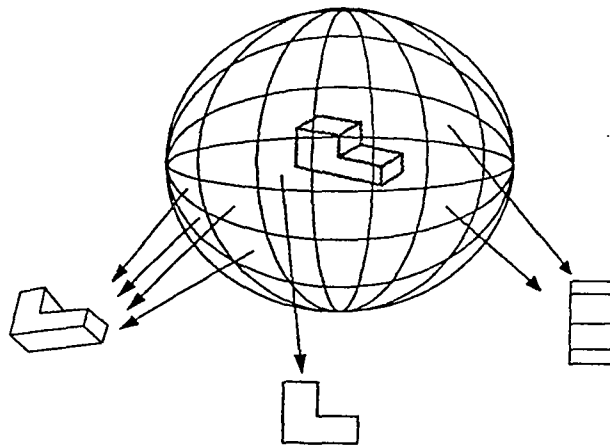


Figure 6.2 - 3D Viewing Space with an Example of L-shaped Object

The aspect graph is one of the most popular viewer-centered representations and often considered to have great potential in computer vision application. The aspect graph of an object is a graph structure in which:

- each node represents a general view of the object as seen from some maximal, connected cell of viewpoint space;
- each arc represents an *visual event* (topologically different views or new features) occurring on the boundary between two cells of the general viewpoint;
- there is a node for each possible general view of the object, and
- there is an arc for each possible visual event [Faug92].

In the last few years, algorithms have been developed to automatically compute the aspect graphs of polyhedrons [GigM90][SteB90][SteB88], general curved objects [EggB90] and even objects with articulated connections between parts [SaSB90]. However, much of the work in this area has a somewhat theoretical flavor and it is unclear that whether aspect graph representation, at least at its present stage, will find practical application. There are three important reasons that this approach has not been heavily used so far in the computer vision community: 1) it requires a large amount of storage space; 2) the matching process is extremely time-consuming because it needs not only search an object model (aspect graph) but also find an isomorphic view (aspect) of the model; and 3) an unique object match is not guaranteed.

A weighted aspect graph approach based on the observable probabilities of features has been proposed [Bena90]. The probability of a feature is defined as the ratio between its viewing region to the total observable area of the viewing sphere. This probability model organizes the conventional aspect graph for the purpose of easier searching but does not reduce the size of the model base. The recognition process still relies on graph-matching techniques. A symmetry (redundancy) detection and saliency (rarity) computational approach has been proposed [Flynn92] for reducing the size of object-centered model base. This is an unnecessary process since the number of objects in a model base is usually not too large (although they will usually have a large amount of 2D views for viewer-centered object model).

In conventional viewer-centered approach, a 3D object is represented by many 2D projections (*views*). This is because, regardless of the significance and global redundancy, they usually store a view that is topologically different from its neighboring views. Any view of the object can then be matched directly with the views stored in the model base by any graph-matching method. It takes $O(m^n)$ amount of computation cost for two graph to match, where m and n are the number of nodes of the *model* and *input graphs*

respectively. As a result, this method of model storage and matching suffers from excessive storage and computational requirements. In addition, only a few of the 2D views in the traditional viewer-centered object model are informative and hence useful in the matching process. Given a view of an unknown object, the cost of searching for the corresponding object in a large model base becomes exceedingly high, even for fairly simple objects.

6.3 PRELIMINARIES

As many terminologies used in previous work have not been unified, in this section, some definitions of terminologies that may be useful throughout this chapter is presented as follows.

<Definition 6.1> A *view*¹ is a 2D projection of a 3D object, i.e. a 2D picture of the 3D object from a particular viewpoint. ■

<Definition 6.2 > An *extreme view*² is a 2D projection of a 3D object when either a surface degenerates precisely into a line or an edge degenerates precisely into a point. In other words, the viewpoint lies directly on one of the object's planes (containing a face) or lines (containing an edge). A few examples of extreme views are given in Figure 6.3. A view which is not extreme is called *non-extreme view*. ■

¹ It is also called a projection or an aspect in other articles.

² It is also called a degenerate view in [KenF87].

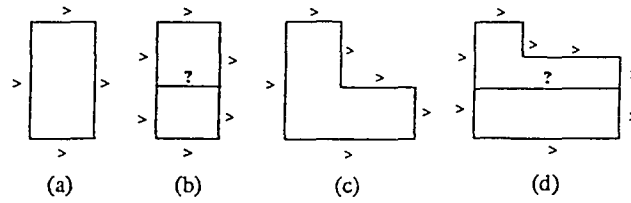


Figure 6.3 - Some Extreme Views of the Object in Figure 6.1(a)

Some neighboring extreme views may form a stable *extreme viewing region* where no new faces, edges or junctions will appear unless the viewpoint shifts slightly across its boundary (sometimes called a visual event). These views are extreme because much information is occluded by the front (visible) faces. Since they are less informative views, many redundant views may often exist among objects and result in an overly large object model. For example the two different objects in Figure 6.1, have exactly the same extreme views seen from many different viewpoints. From a labeling point of view, these extreme views do not always have a unique labeling result. For example, the central horizontal lines shown in Figure 6.3(b) and 6.3(d) can be labeled as "+", "-" or ">", because both ends are T-junctions.

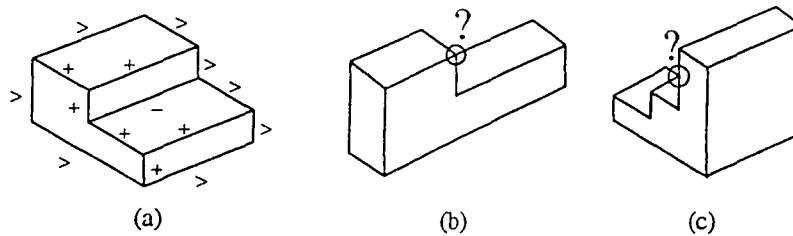


Figure 6.4 - (a) Labelable View (b)-(c) Accidental (Non-labelable) Views

<Definition 6.3> A *labelable view* is a non-extreme view in which all lines have valid or consistent Huffman-Clowes labels. ■

Figure 6.4(a) shows a labelable view. A detail definition of Huffman-Clowes labeling scheme can be found in Chapter 5 of this dissertation.

<Definition 6.4> An *accidental view* [BowD90] is a view where 1) a vertex of an object and some points on an edge of the object project onto the same junction in a view, as in Figure 6.4(b); or 2) points on three different edges of the object project to the same junction in a view, as in Figure 6.4(c). Accidental views are not labelable. ■

6.4 VALID-VIEW OBJECT MODELING BY LABELING

It is assumed that each scene may only consist of a single trihedral object in 3D space. A trihedral object is a closed, bounded and regular subset of R^3 whose boundaries are a set of plane surfaces. In the trihedral object world, each vertex of the object is the intersection of exactly three surfaces and each edge is the intersection of two surfaces. The objects defined in this trihedral object model are also restricted to have *solid* and *opaque* surfaces. Besides, the projection used in this model must be *orthographic* with viewpoint at infinity. However, it could be extended to cover perspective views without any modification. In order to obtain a line-drawing, lines and junctions are first extracted from a 2D intensity image by the edge detector and line tracer. Since many satisfactory approaches have been proposed to produce near-perfect line-drawings [GuHu87] [LieC90] [Huan93], one may assume that all of inputs are perfect line-drawings.

6.4.1 Object Modeling Process

In order to eliminate those extreme and accidental views from the 3D object model base, a method has been developed to distinguish those views from the normal or non-extreme views.

6.4.1.1 How to Remove Accidental Views

The *line-drawing labeling* process detects the accidental views very straightforwardly because the resulting junctions or lines are beyond the Huffman-Clowes labeling domain. In other words, accidental views are non-extreme but unlabelable views. Therefore, the new object model base will exclude those accidental views during the labeling process. A new robust and near-linear-time performance line-drawing labeling approach using the *Cascaded Constrained Resource Planning (CCRP)* model has been proposed in [CheY94a] and presented in chapter 5 of this dissertation. The robustness (labeling lines and junctions simultaneously) and computational efficiency (near-linear-time performance) of this pre-processing step underlies the proposed valid-view 3D object modeling and multi-view matching approach, which uses labeling extensively.

Another essential power of the labeling process is discovered that it can also aid in the detection of imperfect line drawings, such as missing lines or junctions. For example, since one of the lines is missing in the line-drawing shown in Figure 6.5, no consistent labeling will exist. Referring to Figure 5.3, there is no L-junctions with two "+" labels or one "+" and one "-" labels. Therefore, the labeling process will report "no solution" for this line-drawing.

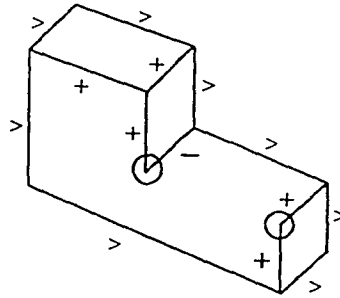


Figure 6.5 - An Imperfect Line-Drawing

6.4.1.2 How to Remove Extreme Views

From the definition, an extreme view appears only when a Y-or W-junction becomes a T-or L-one after a movement. If the objects are right-angle trihedrons, an extreme view is one does not contain any Y-or W-junction. In other words, for right-angle trihedrons, the extreme view is composed only of T-and L-junctions only. In addition, for any arbitrary trihedron, the extreme view is the view without any *Y-W junction pair*. Hence, for any one of above cases, if a view has an Y-W junction pair, it must be a non-extreme view. The following lemma shows that a Y-junction and W-junction always appear together in a single-object scene.

Lemma 6.1: *For any Y-junction in a single-object scene, there is at least one W-neighboring junction. In multiple-object scenes, a W-junction will become a T-junction due to occlusion. ■*

Proof:

For a single-object scene, a Y-junction is formed by the intersection of either two or three faces in the real 3D trihedral object world, as shown in the first two figures of Figure 6.5. If it has been proven that there is at least one W-neighboring junction required for Y-junction with two faces then it can also be proven that there exists at least one W-neighboring junction for three-faces Y-junction. It is noted that ">" represents an arrow

pointing towards the junction and "<" represents an arrow pointing ways from the junction. The visible face is always located at the right-hand side of the arrow. The proof is as follows:

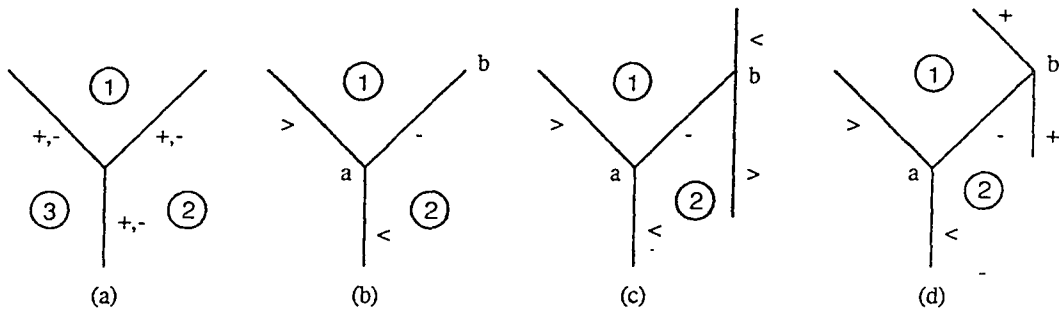


Figure 6.6 - A Y-junction Has At Least One W-Neighboring Junction

The two visible faces of a Y-junction intersecting at an edge $\langle a, b \rangle$ with a "-" label, are bounded and not co-planar. The edge $\langle a, b \rangle$ should not be connected to a L-junction because there are two visible faces. It can only be connected to a T-or W-junction. If $\langle a, b \rangle$ is connected to a T-junction, there will be three visible faces for a T-junction with a "-" stem-branch. In other words, there must exist multiple objects in the scene. This is not allowed in our assumption. Therefore, for a Y-junction with two visible faces only one possible type of neighboring junction can be connected to edges with the "-" label - the W-junction. Their connection is shown in Figure 6.5(d).

Since there is only one possible connection to a Y-junction with two faces, it can be concluded that there exists at least one W-neighboring junction for a Y-junction in the trihedral object model. **Q.E.D.**

6.4.1.3 Size of Model Base

The maximal number of labelable views for general trihedral objects in the orthographic viewing space can be estimated as follows. Let the number of faces, edges, and vertices of a general trihedral objects be $O(f)$, $O(e)$ and $O(v)$ respectively. From Euler's formula: $v - e + f = 2$, which is true for any spherical polyhedron where v is the number of vertices, e the number of edges and f the number of faces. It is obvious that e is always greater than v or f ; and v is always greater than or equal to f (where, $f \geq 4$, $e \geq 6$, $v \geq 4$. The equalities hold only for the most simple tetrahedron). However, they are linearly dependent on each other, i.e. $f \approx e \approx v$. The following are approximations of the number of extreme, non-extreme, and accidental views for general trihedral objects.

1. The approximate number of *accidental views* is $O(v) + O(e^3)$.

- $O(ev)$ is the number of views in which a vertex of an object projects onto an edge of the object.
- $O(e^3)$ is the number of views in which three different edges project onto the same vertex.

(In real cases, the number of accidental views is much less than this number. As a matter of fact, the number of accidental views will usually approach zero.)

2. The approximate number of *extreme views* is $O(f) + O(f^2)$.

- $O(f)$ is the upper bound on the number of views with viewing directions perpendicular to the object face.
- $O(f^2)$ is the upper bound on the number of views with viewing directions coplanar to (i.e. lying on) the faces of the object. (Since an edge is the intersection of any two faces, $O(f^2)$ approaches $O(e)$ in real cases.)

3. The approximate number of *non-extreme views* is $O(f^3)$

- $O(f^3)$ is the upper bound on the number of views in which the three faces of every vertex are visible (Y-W junction pair). (Since a vertex is the intersection of any three faces, $O(f^3)$ approaches to $O(v)$ in real cases.)

Therefore, the ratio of remaining views, i.e. no-extreme views (note: redundant views have not yet been removed) to the number of views stored in the traditional aspect graph approach in the new model is:

$$\frac{O(v)}{(O(f)+O(e)+O(v))} \approx \frac{1}{3}$$

For example, in Figure 6.6(a) there are 6 faces, 8 vertices and 12 edges for a cube. Thus, the estimated ratio is: $8/26=0.307$, which is equal to the actual ratio. For the tetrahedron, in Figure 6.6(b), there are 4 faces, 4 vertices and 6 edges. In this case, the number of non-extreme views is 4 and the total number of views in the traditional aspect graph approach is 14. There is no accidental views for a tetrahedron. Therefore, the estimated ratio: $4/14$, is also exactly equal to the actual ratio. The estimate and actual ratio for the drawing in Figure 6.8(a) are 0.316 and 0.27 respectively. Similarly, they are 0.318 and 0.26 for the drawing shown in Figure 6.8(b). Obviously, the valid-view model saves at least approximately $2/3$ amount of storage space than the traditional viewer-centered object model. Another large amount of space will be further reduced by eliminating all redundant views during the model building process. However, the number of redundant views cannot be estimated precisely because it is highly dependent on the object geometry. In other words, an object will have more redundant views if it has more symmetrical views.

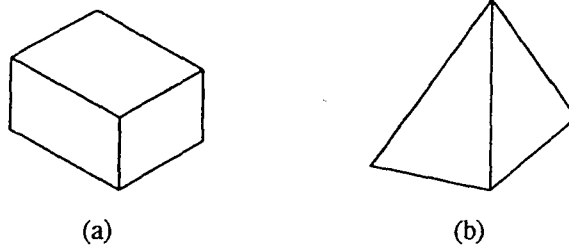


Figure 6.7 - (a) $f = 6, v = 8, e = 12$ (Cube); (b) $f = 4, v = 4, e = 6$ (Tetrahedron)

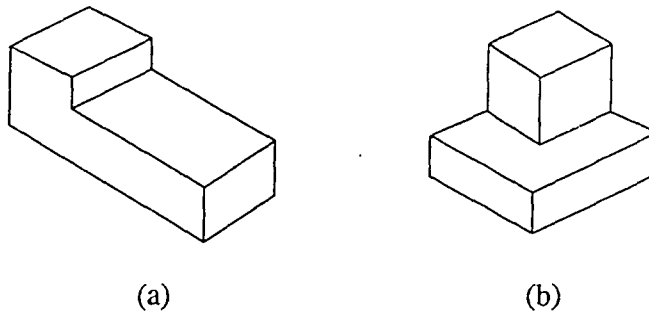


Figure 6.8 - (a) $f = 8, v = 12, e = 18$ (L-shaped); (b) $f = 9, v = 14, e = 21$

6.4.1.4 How to Remove Redundant Views

Before adding labelable views into a model base, their feature vectors and textual representation must be defined. In order to reduce the object model size, the redundant views must be detected and removed during the model building process, especially when the objects are highly symmetrical.

<Definition 6.5> A labelable view can be represented as an 8-tuple *feature vector*, $\langle F,$

$P, M, A, Y, W, T, L \rangle$, where:

F is the number of visible faces;

P is the number of Plus labels;

M is the number of Minus labels;

A is the number of Arrow labels;

Y is the number of Y-junctions;

W is the number of W-junctions;
 T is the number of T-junctions; and
 L is the number of L-junctions. ■

The number of visible faces is obtained by the linear-time surface extraction algorithm mentioned in chapter 4. After pre-processing and labeling, it is very easy to obtain other features. These eight features are invariant to translations, scalings and rotations. Using hashing mechanism, these eight features are encoded into a number, which is called the signature of the input view.

<Definition 6.6> Let $\langle f_1, f_2, f_3, \dots, f_8 \rangle$ be the feature vector of a labelable view i , the corresponding *signature*, S_i , is defined as: $S_i = \sum_{j=1}^8 f_j \cdot 2^{8-j}$. ■

Although this signature representation is designed for hashing, it is not unique (but could have been made unique) and has a property that higher dimensional feature contributes more significantly to the value of S_i . On the other hand, a view is said to be more informative if S_i possesses a larger value. The choice of 2 as the basis is just for sake of programming simplicity. To have a more distinguishable value of the signature, the basis can be set as: $\max(f_i) + 1$ for each view.

<Definition 6.7> A *valid view* of an object to be stored in our object model is a view that satisfies the following three conditions: 1) it is a non-extreme view; 2) it is a labelable view; and 3) it has a distinct signature with respect to the object. The model-building process is shown in Figure 6.9.

As a matter of fact, it is often difficult to estimate the savings in storage of this new modeling method because the third reduction totally depends on how symmetrical the

model object is. The object model only stores the views with different signatures for each individual object. Views with equal signatures but from different objects will be hashed into the same address and chained along the same list. Since extreme views have been eliminated, the length of each chain will not grow too much to affect the matching efficiency.

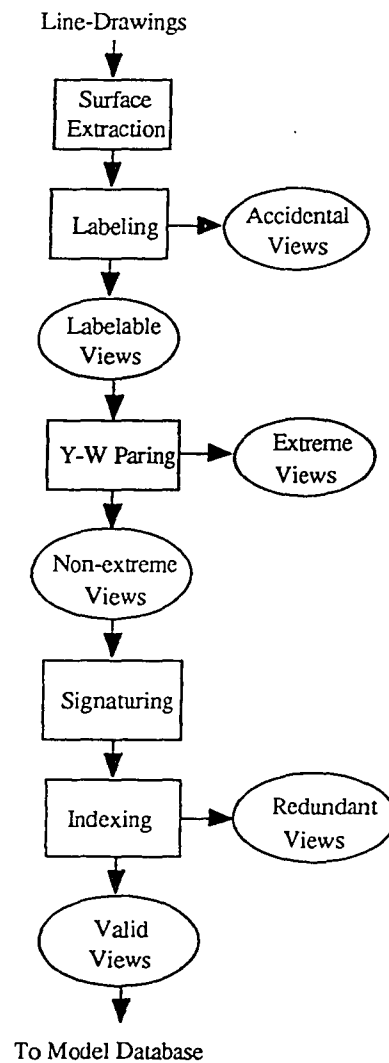


Figure 6.9 - Valid-View Object Modeling Process

6.4.1.5 An Upper Bound on the Number of Valid Views for Right-Angle Trihedrons

In this section some proofs of the upper bound on the number of valid and extreme views for right-angle trihedrons with either convex or concave shapes will be shown.

<Definition 6.8> An *equivalent-face pair* has two sets of faces that occlude each other in the opposite viewing directions. In other words, we can only see all faces of one set of the pair at a time. ■

The maximum number of visible faces mvf , of an object can be obtained from the following two steps: 1) find all equivalent-face pairs, ef_i ; 2) $mvf = \sum_{ef_i} \max(\# \text{ faces})$ for all equivalent-face pairs, ef_i , of the object. For example:

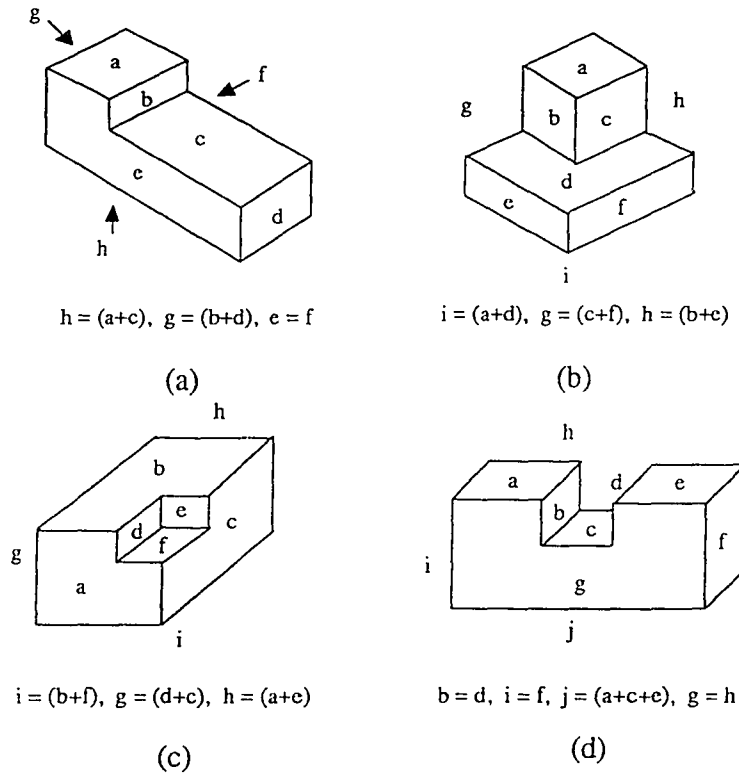


Figure 6.10 - Right-angle Trihedrons

In the above figure, Figure 6.10(a) has 5 at most visible faces and Figures 6.10(b), (c) and (d) all have 6 visible faces from the given viewing angles. So, the maximum number of visible faces for each object equals to the sum of the number of faces on the right hand side which is the maximum number of faces of the equivalent-face pair. Then, the following lemma and corollary are resulted:

Lemma 6.2: *For an object the upper bound on the number of valid views equals to $\#V + \#$ all possible combination of ef_i , where $\#V$ is the number of vertices and the number of visible faces of all possible combinations ranges from 3 to mvf . ■*

Proof:

For non-extreme views, $\#V$ represents the upper bound of the number of views that have *three* visible faces. Some of the combinations of ef_i do not exist in reality. For example, (i, g, a, e) in Figure 6.10(c) will not be a possible combination because face e is not connected to the other three faces. In addition, since symmetricity is not checked at this moment, redundant views might be included, if any. Therefore, in reality, the number of valid views will be less than this bound. As this is the only bound which can be estimated systematically, one can claim that it is the only upper bound on the number of valid views so far. Q.E.D.

Corollary 6.1: *The upper bound on the number of extreme views of an object is equal to $\#F + \#E$, where $\#F$ is the number of surfaces and $\#E$ is the number of edges of the given 3D object. ■*

Proof:

By definition, faces and edges in extreme views are degenerated to edges and vertices respectively. Therefore, the viewing sphere has at most $\#F + \#E$ extreme views because some of the faces will be degenerated to edges with respect to that view. Similarly, some of the edges will be degenerated to vertices. For example, the number of extreme views for a cube is 18. Q.E.D

Figure 6.10(a) shows an example with mvf 5, the upper bound of the number of valid views are enumerated by the following procedures:

Case 1: (The number of visible faces is 5) there are two possible combinations:

(a,c,b,d,e) and (a,c,b,d,f);

Case 2: (The number of visible faces is 4) there are four possible combinations: (a,c,g,e),

(a,c,g,f), (h,b,d,e) and (h,b,d,f);

Case 3: (The number of visible faces is 3) it is equal to the number of vertices, $\#V=12$.

Thus, the upper bound on the number of valid views is 18 (12+2+4) and the upper bound on the number of extreme views is 26 (8+18). So, the upper bound on all possible views then becomes 44 (18 + 26). However, in [EdOS83], the upper bound is $C(f,3) + C(f,2) + C(f,1) = (f^3 + 5f)/6 = 92$ and, in [Watts88], the upper bound is $4 + 2C(f,2) - 2 = f^2 - f + 2 = 58$.

Besides, the upper bound computation is only applicable for convex polyhedrons in both [EdOs83] and [Watts88]. In proposed actual implementations, the number of views stored for the L-shaped object are: 71 views in [WanF90] and 57 views in [ChaH92] respectively.

Knowing the upper bound of the number of valid views can help us to determine the size of model base ahead of time. Our upper bound estimation model is only applicable

for so-called right-angle trihedrons in which object surfaces are either parallel or perpendicular to each other. Another assumption have been made in this model is that only orthographic views are considered with viewpoints at infinity and perspective viewing cases are excluded.

6.4.2 Canonical View List

Two objects are topologically different if they have different singularity configuration in 3D space. On the other hand, if two objects are distinct, they must be topologically different. Hence if two objects are topologically different, they must have one "special view" to distinguish the corresponding objects. This view must be one which is labelable and non-extreme. However, two objects having the same topological (3D) structure do not necessary have the same set of views. The following lemmas and proposition are useful in supporting the concept of canonical views for 3D objects..

Lemma 6.3: *Any two topologically equivalent 2D projections (views) of a 3D object must have the same signature. However, the converse is not true. ■*

Proof:

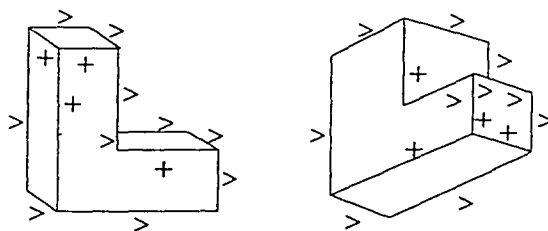
Any two 2D projections of a 3D object is said to be topologically equivalent if and only if:

- (1) they have same number of visible faces, edges and vertices; and
- (2) their topological representations (graphs) are isomorphic; i.e. their *edge function* and *vertex function* are one-to-one and onto.

Let the feature vectors of the corresponding signatures of the 2D projections be $\langle f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8 \rangle$ and $\langle f_1', f_2', f_3', f_4', f_5', f_6', f_7', f_8' \rangle$, respectively. From (1) one can confirm that $f_1 = f_1', f_2 + f_3 + f_4 = f_2' + f_3' + f_4'$, and $f_5 + f_6 + f_7 + f_8 = f_5' + f_6' + f_7' + f_8'$. And from

(2) one can further confirm that their feature vectors are equivalent. Hence they produce the same signature individually.

Conversely, the following example, shown in Figure 6.11, can be used to prove it. It is obvious that the following two projections have different topologies but the same signature. Q.E.D.



Feature Vector = $\langle 4, 4, 0, 10, 1, 4, 1, 5 \rangle$
 Signature = 959

Figure 6.11 - Two Topologically Different Views With Equivalent Signature

Corollary 6.2: Two neighboring projections within an open neighborhood of viewpoints (of any general viewpoint, i.e. an aspect), in the viewing space are topologically equivalent if and only if they have the same signature. ■
(All projections of an aspect may have different metric properties but equivalent graph structures and topologies.)

Proof:

From the definition of an aspect, since all views in the aspect are topologically equivalent, they must have the same signature. Furthermore, because no visual event will happen when the viewpoint moves from one position to another within the aspect, the number of faces will not change, neither will each edge and junction type. The only differences are the length of the edges and the size of the angles between two edges of the

junctions. Therefore, for all views in an aspect, they are topologically equivalent if and only if they have same signature. **Q.E.D.**

In addition, the larger the signature value of an object view, the more information (features) it can carry and the higher canonicity it is capable of representing. Although the signature representation may not be unique for individual views, a set of valid views must be unique in the model; otherwise, the object will be rejected from the model base because it does not have distinguishable features. The canonical view list of an object model can then be obtained gradually from the following procedure:

Initially, let the canonical view, CV_i of an object i be one of the valid views with the largest signature value, i.e. $CV_i = \max(\text{Signature}(V_{ij})), j=1, i_k$, where i_k is the number of valid views of object i . If CV_i is not unique among the model objects (i.e., there is another object with the same view), append the view with the second largest signature value to the canonical view list. Repeat this procedure during the model building process until the view list is unique to the model. Hence, the canonical view list may contain more than one view and the list should be unique to the model according to the above building process. Therefore, we have the following proposition.

Proposition 6.1: *A canonical view list is the most compact unique representation for a 3D object. ■*

Proof:

According to the definition of an object model base, all stored objects should have distinct features. As a 3D object model base in a multi-view representation is composed of non-redundant (distinct) valid views and the uniqueness has already been verified during the construction of the canonical view list, this list is then unique in the model base. Instead of using all valid views as canonical view list, the list only contains the smallest

number of valid views that can be uniquely identified. Therefore, this canonical view list is most compact unique representation of a 3D object. **Q.E.D.**

6.5 MULTI-VIEW OBJECT MATCHING BY INDEXING

6.5.1 Approach Description

Model indexing approach [CleJ91] is defined as follow: From a given set of features of an unknown object view, rapidly extract a list of objects containing a representative view. Since the new model base is developed by hashing all the signatures of valid views into a chained hash table, the matching process consists of computing the address of the unknown object view to determine its existence in the hash table. Usually, it can be done in constant time. However, for a single input view, the system may not be able to find an exactly-matched object from the model base if the view is not unique. Therefore, for the purpose of reliable recognition, humans perceive a 3D object usually by looking for a set of different but informative views from different viewpoints. The algorithm is shown below.

-
1. Capture a new view;
 2. Extract features:
 - 2.1 Surface extraction;
 - 2.2 Line/Junction labeling;
 - 2.3 Y-W paring;
 - 2.4 Signature Generation;
 3. Indexing to model base;
 4. IF no match THEN stop and exit;

ELSE IF an unique view has been found THEN report the matched object ID and exit;
ELSE IF enough confidence THEN report the matched object ID and exit;
ELSE IF need more views THEN go to step 1;
ELSE report a list of object ID and exit;

As a result, the matching process stops only when 1) there is a unique input view (only one object contains such view); 2) there is a unique sequence of input views (only one object has such a sequence of input views); or 3) there is no match at all. The confidence or uniqueness of the candidate object increases with more captured views.

6.5.2 A New Confidence Computation Model

In this section, a new confidence computation model for the multi-view indexing object recognition system will be presented. Since the significance of each input view is different (the significance is measured by the value of the signature, computed from the 8-tuple feature vector), the confidence of an object is accumulated or calculated by the weights (signatures) from the input views. Views possessing higher signature values will contribute more confidence to the objects containing them. In addition, the sequence of input views will also affect the final confidence of this new model. The earlier in the process which a view with a large object signature is captured, the higher the obtained confidence value will be.

For example, Figure 6.12, a model base has 10 objects, where obj_1 has a view list (1231, 959, 942, 729, ...), obj_3 has views (942, 729, ...), obj_6 has valid views (959, ...) and obj_{10} has (729,...) views, respectively.

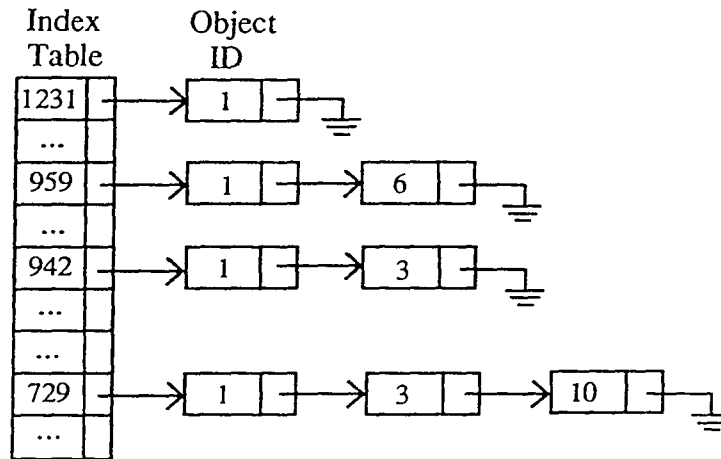


Figure 6.12 - Example of a Model Base

6.5.2.1 Rules of Confidence Combination

Since the signature value carries the canonicity (significance) information of each input view, the larger the signature value, the bigger contribution it will make to the final confidence value. Therefore, a weighted confidence computation model has been developed to cope with the signature value for computing the confidence of any object k , obj_k :

$$C_{com} = \frac{1}{\max(S_a, S_i)} [S_a * C_a(obj_k) + S_i * C_i(obj_k) - \min(S_a, S_i) * C_a(obj_k) * C_i(obj_k)] \quad (6.1)$$

$$\text{and } S_{com} = \frac{(S_i + S_a)}{2} \quad (6.2)$$

where, S_i and S_a are the signature values for input and accumulated view respectively. The signature of the accumulated view is the combined signature from among all previously captured views, calculated according to equation (6.2). The result of the weighted confidence computation model for the model base shown in Figure 6.12 is:

Table 6.1 - The Accumulated Confidence Table

	<i>obj₁</i>	<i>obj₂</i>	<i>obj₃</i>	<i>obj₄</i>	<i>obj₅</i>	<i>obj₆</i>	<i>obj₇</i>	<i>obj₈</i>	<i>obj₉</i>	<i>obj₁₀</i>
view 729	0.33	0	0.33	0	0	0	0	0	0	0.33
view 942	0.63	0	0.63	0	0	0	0	0	0	0.33
view 959	0.77	0	0.63	0	0	0.5	0	0	0	0.33

* The input view sequence is (729, 942, 959).

6.5.2.2 Advantages of the Model

1. The resulting confidence value is always between 0 and 1.
2. The confidence-combining rule includes the significance value of each input view, since each view has different significance.
3. The final confidence value will become 1 when a canonical view is captured.
4. The earlier in the process in which an important view has been captured, the higher the obtained confidence value will be.

The following two lemmas prove advantages 1 and 3 above.

Lemma 6.4: *The combined confidence is always less than one in the weighted confidence model.*

proof:

Let $S_i = \max(S_a, S_i)$, then the comined C_{com} equals:

$$\begin{aligned}
 C_{com} &= \frac{1}{S_i} (S_a * C_a + S_i * C_i - S_i * C_a * C_i) \\
 &= C_i + \frac{S_a}{S_i} C_a - \frac{S_a}{S_i} C_i * C_a
 \end{aligned}$$

$$= 1 - (1 - C_i) \left(1 - \frac{S_a}{S_i} C_a\right).$$

Since $S_i > S_a$ and C_i and C_a are all positive real numbers, the combined C_{com} will be lesser or equal to 1. The proof is the same for $S_a = \max(S_a, S_i)$. **Q.E.D.**

Lemma 6.5: *When a canonical view of an object is obtained, its corresponding confidence value will be 1.*

Proof:

Since a canonical view is defined as an unique view with the largest signature value, $S_i = \max(S_i, S_a)$ and $C_i = 1$. The combined confidence C_{com} is:

$$\frac{1}{S_i} (S_i * 1 + S_a * C_a - S_a * C_a) = 1 \quad \text{Q.E.D.}$$

6.5.2.3 Influence of Input Sequence

The following four examples show how the sequence of input views affect the final confidence value.

Table 6.2 - The Accumulated Confidence Table for Different Viewing Sequences

Sequence	obj1	obj2	obj3	obj4	obj5	obj6	obj7	obj8	obj9	obj10
(729,942,959)	0.77	0	0.63	0	0	0.5	0	0	0	0.33
(729,959,942)	0.78	0	0.62	0	0	0.5	0	0	0	0.33
(959,942,729)	0.81	0	0.63	0	0	0.5	0	0	0	0.33
(729,1231)	1.0	0	0.33	0	0	0	0	0	0	0.33

It is assumed that S_a and C_a are the signature and confidence values of the accumulated view and S_i and C_i are the signature and confidence values of the new input

view. Then the following lemma is to support this confidence-computation model to raise practical interest and importance.

Lemma 6.6: *The combined confidence value of the weighted confidence model is always monotonically increasing when the accumulated view has a larger or equivalent signature (importance) than the input view. On the other hand, for a larger signature (less important) input view, the combined confidence will increase only when the confidence of input view (C_i) and the confidence of the accumulated view (C_a) satisfies the following relation:*

$$C_i > \frac{(1 - \frac{S_a}{S_i})C_a}{1 - \frac{S_a}{S_i}C_a},$$

where S_i and S_a are the signatures of the input and accumulated views respectively.

proof:

(1) If $S_a = \max(S_i, S_a)$, the combined confidence, C_{com} , is always monotonically increasing.

$$\begin{aligned} \frac{1}{S_a}(S_a * C_a + S_i * C_i - S_i * C_a * C_i) - C_a &> 0 \\ \Rightarrow \frac{S_i}{S_a} * C_i * (1 - C_a) &> 0 \end{aligned}$$

Since S_i/S_a , C_i and $1 - C_a$ are all greater than 0, C_{com} is always greater than C_a .

(2) If $S_i = S_a$, the confidence value is definitely monotonically increasing.

(3) On the other hand, if $S_i = \max(S_i, S_a)$, C_{com} is monotonically increasing only if:

$$\frac{1}{S_i}(S_i * C_i + S_a * C_a - S_a * C_i * C_a) - C_a > 0$$

$$C_i > \frac{(1 - \frac{S_a}{S_i})C_a}{1 - \frac{S_a}{S_i}C_a} \quad \text{Q.E.D.}$$

The above lemma shows that the earlier the more important view is captured for an object, the higher the obtained confidence and the faster the convergence will be.

6.6 EXPERIMENTAL RESULTS

A simulation system called 3DOMMS (3D Object Modeling and Matching System) has been built to prove the efficiency of this new labeling-based valid-view modeling and multi-view matching approach. The detail functions of this system and a walk-through example can be found in Appendix A. The canonical views and valid views of tested model objects and their corresponding statistical information are given in Appendix B. It is obvious that this modeling approach requires few valid views since other uninformative views have already been rejected during the model building process. For instance, obj1 only has 4 valid views (Figure 6.13) and obj2 requires 13 valid views. However, in [WanF90], obj1 and obj2 need 71 and 122 views respectively.

Those model objects are created by a 3D graphic tool called SHOWCASE in graphic supercomputer ONYX system. The all tested views are captured and drawn by 3DOMMS one-by-one for modeling.

This simulation system is implemented in C++ language on a 486-PC. Users can construct drawings on the screen directly or import line-drawings via an input file. Imported drawings were created by either the graphical drawing subsystem or an image processor. A browser function is provided to users for examining objects already stored in

the object model base. An object number will be displayed on the message window indicating the total number of objects that have been modeled. The user should input a new object number corresponding to each input line-drawing during the modeling process. For a low confidence (less important) view, the system will ask for input more views.

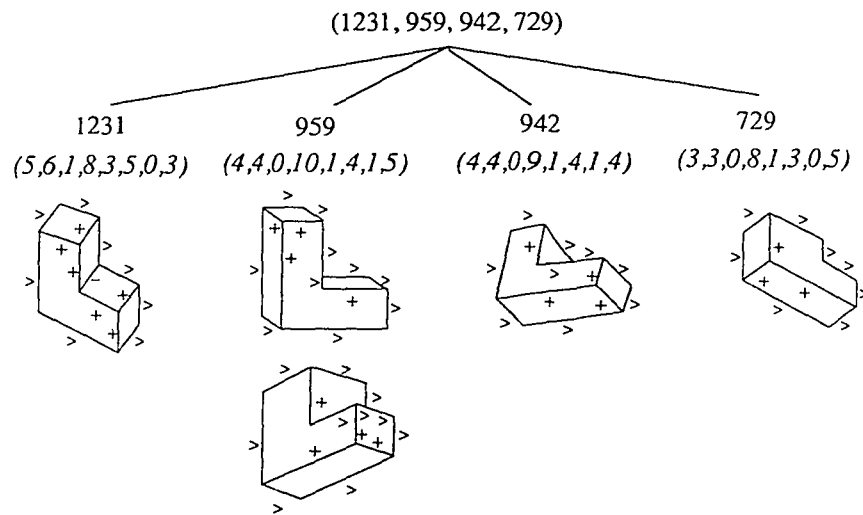


Figure 6.13 - Valid Views of L-shaped Object

The performance of the simulated 3D object recognition system is shown in Figure 6.14. It is noted that the x-axis is labeled by the summation of #E (number of edges), #J (number of junctions), and #S (number of surfaces), since these are the basis for signature generation. After the signature is generated, the matching is accomplished in a constant time.

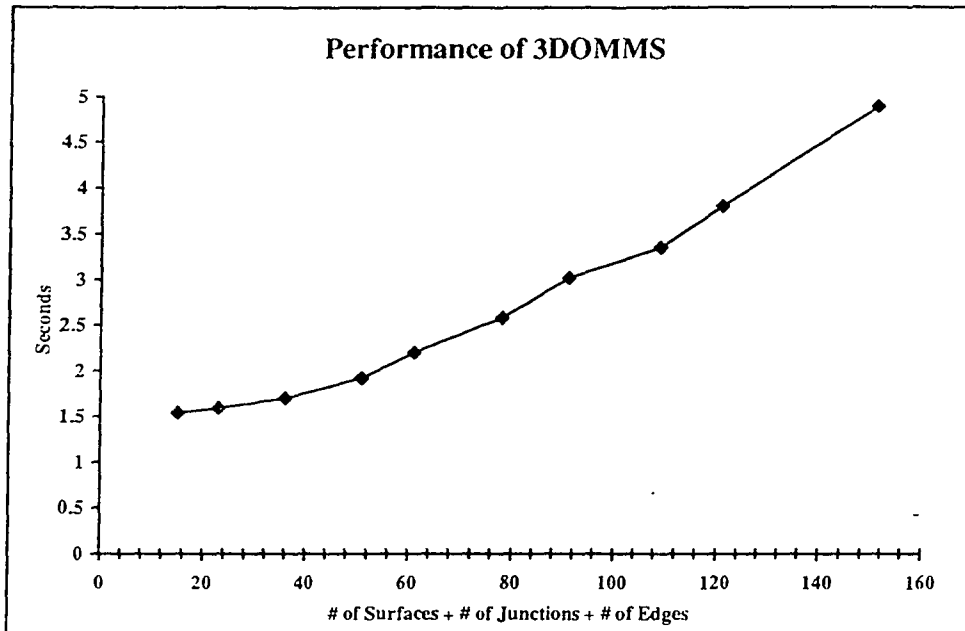


Figure 6.14 - Performance of the Simulated 3D Object Recognition System

6.7 SUMMARY

Using as little storage space as possible and achieving fast and reliable object matching are two crucial goals for the realization of a real-time 3D model-based object recognition system. A novel 3D object modeling and matching approach based on object labels of its 2D line-drawings has been developed and its ability to achieve these goals has been proven in this chapter. With the effectiveness of our labeling process, the object's *canonical view list* and *automatic multi-view indexing mechanism* are proposed and tested fully utilizing labels of lines and junctions. This new approach provides not only faster but also a more natural and reliable perception process over the traditional single-view approach. For curved objects, one additional label is required to represent limbs where the surface curves smoothly around to occlude itself [Mali87]. The labels of C-junction, where the labels will change from convex to occluded or concave along the same curve edge [Coop93]. A

complete labeling dictionary for curved line-drawings needs to be developed and using the same CCRP-based labeling algorithm curved objects are expected to be included in future research.

CHAPTER 7

CONCLUSIONS

In the previous chapters, a set of labeling, modeling and recognition algorithms, models and systems for both 2D and 3D line-drawing interpretation were successfully developed. The motivations and approach overviews of this research were discussed in Chapter 1. In Part I, Chapter 2 and 3 presented three major components, i.e. labeling, modeling and recognition, which were developed via a set of proposed approaches for understanding 2D line-drawing images. Chapter 2 mainly described image abstraction and blurring approaches for the segmenting symbols from electrical engineering drawings. Chapter 3 presented a hierarchical neural network approach for successful recognition of 34 different types of logic gates. In Part II, Chapter 4, 5, and 6 presented a set of new labeling, modeling and recognition approaches for the interpretation of 3D line-drawings. In chapter 4, a new surface extraction algorithm was proposed which is capable of extracting surfaces from a given projection of any trihedron in a linear time. In chapter 5, a robust and efficient line-drawing labeling algorithm that labels lines and junctions alternately was presented. Its near-linear time complexity, with respect to the number of junctions and edges, motivated the extensive usage of the labeling process for 3D object modeling and recognition. Chapter 6 has developed a labeling-based valid-view object modeling and multi-view matching (recognition) approach which validated the advantages of extensively use of symbolic features for the line-drawing interpretation applications. In that chapter, some proofs and experimental results demonstrated the advantages of these approaches over other traditional object modeling and matching methods. An experimental software, 3DOMMS, has been implemented to simulate and prove that these newly proposed approaches work well. This chapter summarizes the contributions of the

overall research work covered in this dissertation and lays out suggestions for future enhancements.

7.1 CONTRIBUTIONS

The major contributions made in this dissertation are:

(1) *Development and demonstration of a general paradigm for solving the line-drawing interpretation problem:*

Three common and major components for solving both 2D and 3D line-drawing image interpretation problem were first proposed in this dissertation. They are namely: *labeling*, *modeling* and *recognition*. The labeling process involves feature extraction and grouping. The output of the *labeling module* is a set of features known as symbolic labels of the corresponding objects. The *modeling module* automatically encodes these object features into the model base. The input to the *recognition module* is just the set of symbolic labels, however, the module performs recognition by matching the input labels to the encoded object labels in the model base. These three modules are applicable to understanding both 2D and 3D line-drawings for different application domains.

(2) *Symbol segmentation via image abstraction and blurring:*

An automatic 2D object (symbol) segmentation (labeling) approach for the electrical engineering drawing image has been developed. Utilizing image abstraction and blurring this approach has been proven to be capable of identifying both open- and close-loop symbols of various sizes in the experiments. This approach segments symbols in a global and fuzzy way and hence is applicable to many different types of

drawings. Instead of thinning and vectorizing the given line-drawing images, symbols can be segmented in 2D space. As there is no need for data interchange, data parallelization can be performed to speed up at each processing step.

(3) *Hierarchical neural network for symbol modeling and recognition:*

A set of translation-, scaling- and rotation-invariant moment features were designated for symbol modeling and recognition. A hierarchical neural network approach for symbol modeling and recognition was proposed in this research to achieve the goals of *incremental extension* and both *minimal human involvement and storage*. Faster convergence and higher recognition rates have been clearly demonstrated by the experiments.

(4) *Linear-time surface extraction algorithm:*

Unlike previous approaches, a polygon-division-based approach to iteratively divide the object's exterior contour into a set of minimal regions, which correspond to visible surfaces of the 3D objects in the scene, is proposed. The easy implementation and lack of the need to perform angular computation, sorting and pseudo surface generation are its advantages over others. In addition, this algorithm has an automatic hole-detection capability, which is not available in any other algorithms.

(5) *Line and junction labeling by a Cascaded Constrained Resource Planning model:*

A robust and efficient 3D line-drawing labeling approach has been proposed and implemented. Traditional line-drawing labeling approaches only label lines under the assumption that junction types are known in advance, whereas this new approach labels lines and junctions alternatively and is developed under a Cascaded Constrained Resource Planning (CRP) framework. This heuristical algorithm has been

demonstrated to have near-linear-time performance (with respect to the number of edges and junctions). No backtracking is needed for most of the test cases. Its effectiveness proves to enable extensive usage of the labeling process for 3D object modeling and recognition.

(6) *Valid-view object modeling:*

Based upon the labeled line-drawing images, a novel valid-view 3D object modeling approach has been developed. This modeling approach requires less storage and is able to achieve faster retrieval time as only informative valid views need to be stored. More than two-third of the views of traditionally used by other methods can be eliminated from the model base. An upper bound on the number of valid views for right-angle trihedral objects has been proposed and proven. An 8-tuple symbolic features and signature representation facilitates simple-structured and lesser storage of the object model base.

(7) *Multi-view object recognition by symbolic indexing:*

Instead of using the time-consuming graph-matching approach, in this research, a set of invariant features are extracted and the *signature* of each *valid view* is generated for automatic object-matching via symbolic indexing. For reliable object recognition, a multi-view matching strategy by looking for a set of different but informative views at different viewpoints has been developed. A new confidence computation model was also proposed to compute the confidence of a sequence of input views based not only on the significance of individual views but also on their input order. A 3D Object Modeling and Matching System (3DOMMS), which demonstrates the feasibility and merits of all of the proposed approaches, has been implemented for the recognition of trihedrons.

7.2 DIRECTIONS FOR FUTURE RESEARCH

The problems of symbol segmentation, modeling and recognition have been successfully solved in this dissertation. However, a complete ACID system not only has to recognize symbols but also texts associated with the corresponding symbols. Moreover, drawing reconstruction is required to produce a better and accurate output. The directions for future research in 2D line-drawing interpretation are suggested as follows:

(1) *Integration with Optical Character Recognition (OCR) to obtain more knowledge about the drawing:*

It is possible to apply the proposed symbol segmentation and recognition approach for text images. Challenges in this direction include 1) automatic separation of texts and symbols; and 2) understanding the relationship between texts and symbols. Once both type of information and their relationships are obtained, drawing reconstruction will become a reality.

(2) *Ability to handle hand-drawn line-drawings:*

Presently, the symbols are limited to be drawn by template. As the market of pen-based computer grows up rapidly, to segment and recognize symbols from free-hand-drawn line-drawings is a great challenge for this research. Since the proposed segmentation approach does not look into detail structure of symbols, it may be directly applicable for such drawing domain. How to train the symbols (as well as characters) from different written-styles is an important and still unsolved problem. However, it is the most challenging work.

For 3D line-drawings, the following extensions to the proposed approaches that may be applied to many other applications are suggested:

(1) *The ability to handle multi-object scenes and objects with holes:*

The current system does not allow a line-drawing to have multiple objects, shown as Figure 7.1. An object separating algorithm is required to partition multi-object line-drawing into several single objects and then perform recognition individually. The labeled line-drawing and T-type junction are useful clues for segmenting two objects that occlude each other. Since the occluded lines are labeled with arrows, the algorithm could start tracing from a T-type junction, follow the arrow labels, and end up at another T-type junction. More investigation is needed for real implementation.

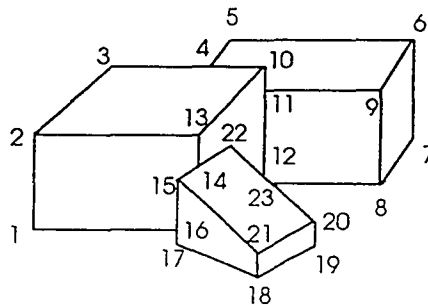


Figure 7.1 - A Multi-object Scene

Another issue of line-drawing labeling requires to be solved in the future is how does system automatically detect holes of the 3D object in the given line-drawing. For example, Figure 7.2 shows 3 different interpretations of the given line-drawing. It can be interpreted as a hole, a floating tetrahedron, or a tetrahedron sitting on a block. To solve this ambiguity, a special processing step which combines range data into line-drawing (intensity data) interpretation is required [GiMA83], since the depth of this isolated sub line-drawings can be derived from the range data. Fortunately, our

surface extraction algorithm can detect those isolated sub line-drawings automatically, if they exist.

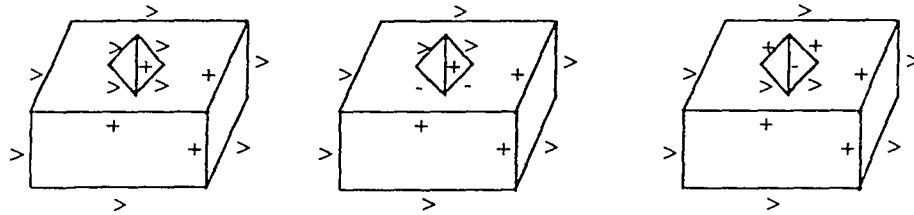


Figure 7.2 - Three Different Interpretations of a Line-drawing

(2) *Fault tolerance capability:*

Since the segmented line-drawing from a multi-object scene is not complete (some portions may be missing due to occlusion), a recognition technique based on partial information can be developed. Moreover, this fault tolerance ability is also required when the input line-drawing is not perfect. Decomposing the entire labeled line-drawing into circuits [GuYH87] may be a potential solution to this problem. Subgraph isomorphism has been proved to have the ability for finding a partial match between a distorted input graph and a model graph [Wong92]. Given a smaller graph (line-drawing), match to a bigger model graph (line-drawing) could be implemented by first decomposing the model graph into a set of smaller subgraph having the same size with the input one and then applying the symbolic indexing approach (proposed in this dissertation) to find the correspondence.

(3) *Planning for next best view:*

The processing order of non-extreme view is a crucial problem for multi-view object matching strategy. A search for the optimal consecutive view will not only minimize camera movement cost but also shorten the recognition process [MavB93].

Since *a priori* geometrical information about the object is not available, the labeled line segments and junctions at the outermost border of the given line-drawing may be useful for the prediction of the next view for object recognition.

(4) *The ability to recognize more objects:*

Beyond the trihedral object world, the results of this research can be easily applied to general polyhedral objects, which include P-junction (peak) K-junction and C-junction (crack) junctions. Some results of curved-objects labeling [Chak79] [Mali87] [Coop93] may be applied to include a larger variety of realistic objects. Figure 7.3 shows some of junction definitions that are beyond the trihedral objects. Label ">>" is called the limb of a curved surface, where it does not correspond to any tangent or curvature discontinuity in the line-drawing. A complete analysis of legal line labelings for general polyhedrons and curved objects is required. Once the dictionaries for general polyhedrons and curved objects is developed, the CCRP-based labeling algorithm can be applied directly. The proposed valid-view modeling and multi-view indexing are still applicable to those extended object worlds, since the symbolic labeling technique remains. In addition, an image pre-processing step is required to detect the curve segments and the junction points, such as curved L-junctions and curved W-junctions.

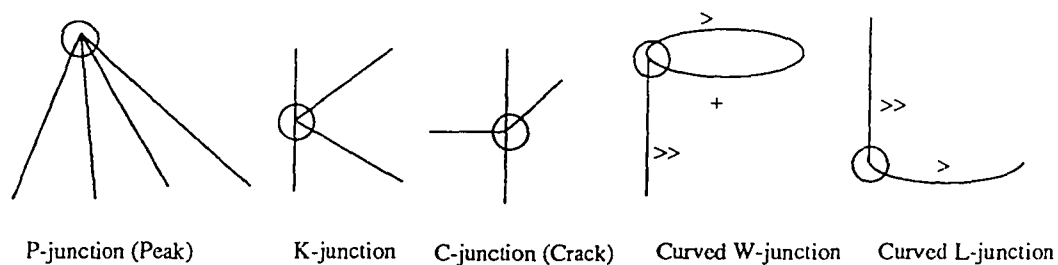


Figure 7.3 - Extended Junction Types for General Polyhedral and Curved Objects

7.3 CONCLUSIONS

To achieve the goal of developing an automatic labeling, modeling and recognition for line-drawing interpretation system, we have proposed a set of efficient approaches that can automatically assign labels to objects. They require less storage for the model base and give faster recognition rates. Image blurring could potentially play a vital role for general object segmentation. Moment invariant features have been successfully used in pattern recognition, however computing invariant moments is too expensive. Since many fast moment computation algorithms or architectures have been proposed recently [Li93][FuYC93], the moment invariant features may be applied to many practical computer vision systems in the near future. To achieve a high recognition rate and incremental model extension, the hierarchical neural network gives the most promising direction. Symbolic labeling, representation and textual matching are no doubt the only way to achieve faster recognition. The most significant contribution of this research is developing a fast symbolic labeling approach which not only solves the bottleneck of traditional labeling problem (low speed) but also opens a door for the applications of labeled line-drawings. Due to the demand for real-time computer vision systems in industrial applications, discovering ways to reduce the size of the model base and speed up the matching as well as searching process are the two main considerations to achieve such a goal. The proposed solutions can realize a truly real-time and geometrically invariant vision system for industry automation of mechanical parts manufacturing. Much work has been done in this research and more extensions are proposed and needs to be solved in the future. In conclusion, this dissertation has laid a new milestone for automatic labeling, modeling and recognition in line-drawing interpretation. Future work in this direction will make the paradigm more mature not only in theoretical research but also in practical computer vision systems.

APPENDIX A

3DOMMS (3D OBJECT MODELING AND MATCHING SYSTEM)

This appendix is divided into four sections. Section A will go through the process of installing 3DOMMS. Section B will show how 3DOMMS's interface works. Section C will list the different commands and describe what each command does. Section D is a walk-through that shows how to add an object into the CV database (canonical view database), how to model an object, and how to match an object. Section E contains a list of files and it shows how the program interacts with the files.

A. Setting up 3DOMMS

Put the diskette in the your disk drive.

Type **a:** or **b:** depending on which drive contains the diskette.

Type **install [drive]** where [drive] is the letter of your hard drive. (E.g., **c:**)

Type **win** to run Windows.

Click on the window where you want your 3DOMMS icon.

Now click on the **File** menu and click on **New...**

Click on **OK**.

Type **3DOMMS**.

Press Tab, and type **c:\3domms\m1.exe** where **c:** is the drive letter of your hard disk.

Press Tab again, and type **c:\3domms** where **c:** is the drive letter of your hard disk.

Now click on **Change Icon...** and choose your icon for 3DOMMS.

Now click on **OK**.

Double-click on the 3DOMMS icon to run 3D Object Modeling/Matching System.

B. Interface

I. Screen Layout

The screen layout contains six windows as shown in **Figure A.1** below.

Title Window - This window always shows "3D OBJECT MODELING/MATCHING SYSTEM," which is the title of this simulation environment.

Main Command Window - This window displays buttons that represent all the major commands used in this system, which are listed and described in section B.

Drawing Window - This window contains a grid area that displays the current drawing.

Browse Window - This is where the canonical view of each object is displayed.

Sub Command Window - This window will show sub-commands when a main command requires further options.

Message Window - This is where messages and user inputs occur.

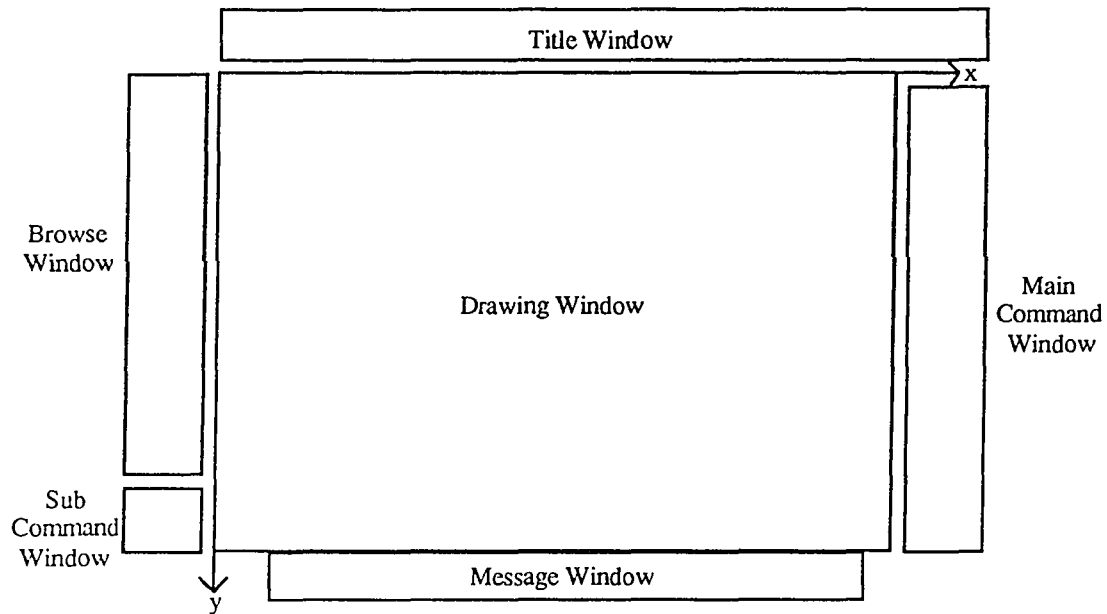


Figure A.1 - Screen Layout

II. Mouse

A mouse is required to run this program. All the commands in the *Main Command Window* and the *Sub Command Window* are activated by a single mouse click. A mouse is also used to input a drawing. The keyboard is only used to type in a filename to save or to read and to confirm a decision.

To avoid confusion, the mouse is restricted to one of three windows at all time. Only the commands inside the current restricted mouse area are needed at that particular moment.

C. Commands

Read - Read drawings from existing DOS files.

Save - Save drawings to a user-specified DOS file.

Input - Allow the user to input a new drawing from scratch.

1. Click on the left mouse button to start a line.
2. A flashing (XOR-mode) line will be drawn on the screen according to the mouse movement.
3. Click on the left mouse button again to replace the flashing line with a permanent line.
4. If an endpoint of the current line is close to an endpoint of an already drawn line, the program will snap the two endpoints together.
5. If an endpoint of the current line is close to a part of an already drawn line, the program is break down the already drawn line and form a T-junction with the new line.

6. Click on the right mouse button to return to the *Main Command Window* when the drawing is complete.

(Note: 3DOMMS cannot recognize a junction with more than 3 lines.)

Edit - Edit current drawing

Insert - Inserting lines

(See *Input* above)

Delete - Deleting lines

1. Click the left mouse button on a line to select a line to be deleted.
2. The selected line will change to a dark red color.
3. More lines can be selected in the same manner.
4. Click on the right mouse button to delete all the selected line and return to the *Edit Sub Command Window*.

Pan - Pan object

1. Click the left mouse button on a node of the object to select that node.
2. To be position the selected node at a different point, click the left mouse button at that point.
3. The selected node will be positioned at that point and the whole object will be positioned at that area accordingly.
4. Click on the right mouse button to return to the *Edit Sub Command Window*.

Return - Return

1. Returns to the *Main Command Window*

Face - Extract faces for the current drawing

1. Faces will only be extracted if the drawing does not have any errors.
2. Errors in the drawing means that some of the lines are not connected.
3. These errors can be corrected using the *Edit* command.

Label - Label the drawing

1. The drawing will only be labeled if faces for the drawing have been successfully extracted.
2. Inconsistency in labeling means that the drawing is not an accepted 3D object.

Browse - Browse the available canonical views

PREVpg - Previous Page

Go to the previous page of canonical views

NEXTpg - Next Page

Go to the next page of canonical views

Return - Return

Returns to the *Main Command Window*

Model - Add or delete the current view from model base

The *Model* command will only activate if the faces of the current drawing have been successfully extracted and the drawing has been successfully labeled. (See *Go Mod*)

Insert - Insert current view into model base

The program will ask the user for the object to which this view should be added.

Delete - Delete current view from model base

The program will ask the user for the object from which this view should be deleted.

Return - Return

Returns to the *Main Command Window*

Go Mod - Add or delete the current view from model base

The *Go Mod* command will activate whenever there is a drawing, but it will not model the object if either *Face* or *Label* is unsuccessful.

Go Mod saves time by going through the whole process of *Face*, *Label*, and *Model* in one step.

The *Go Mod Sub Command Window* contains the same options as the *Model Sub Command Window*. (See *Model* above)

Match - Match to see if the current view is a view of an object in the CV database

The *Match* command will only activate if the faces of the current drawing have been successfully extracted and the drawing has been successfully labeled. (See *Go Mat*)

Go Mat - Match to see if the current view is a view of an object in the CV database

The *Go Mat* command will activate whenever there is a drawing, but it will not match the object if either *Face* or *Label* is unsuccessful.

Go Mat saves time by going through the whole process of *Face*, *Label*, and *Match* in one step.

The *Go Mat Sub Command Window* contains the same options as the *Match Sub Command Window*. (See *Match* above)

C View - Add or delete canonical view of an object to or from the CV database

Insert - Insert current view into CV database

The program will ask the user for the object number of this canonical view.

Delete - Delete a canonical view from the CV database

The program will ask the user for the object number to be deleted.

Exit - Exits 3DOMMS

D. Walk-through

I. Object Manipulation -

At the start of the program, there is no object on the screen. An object has to be either read or created.

Reading an object

Use the left mouse button to click on the *Read* button. Now the program will ask for the filename to be read. Type **obj7.2** and press Enter. The program will now read the file obj7.2. After finish reading obj7.2, the object saved in the file obj7.2 will be printed on the screen as shown in **Figure A.2** below.

3D OBJECT MODELING/MATCHING SYSTEM

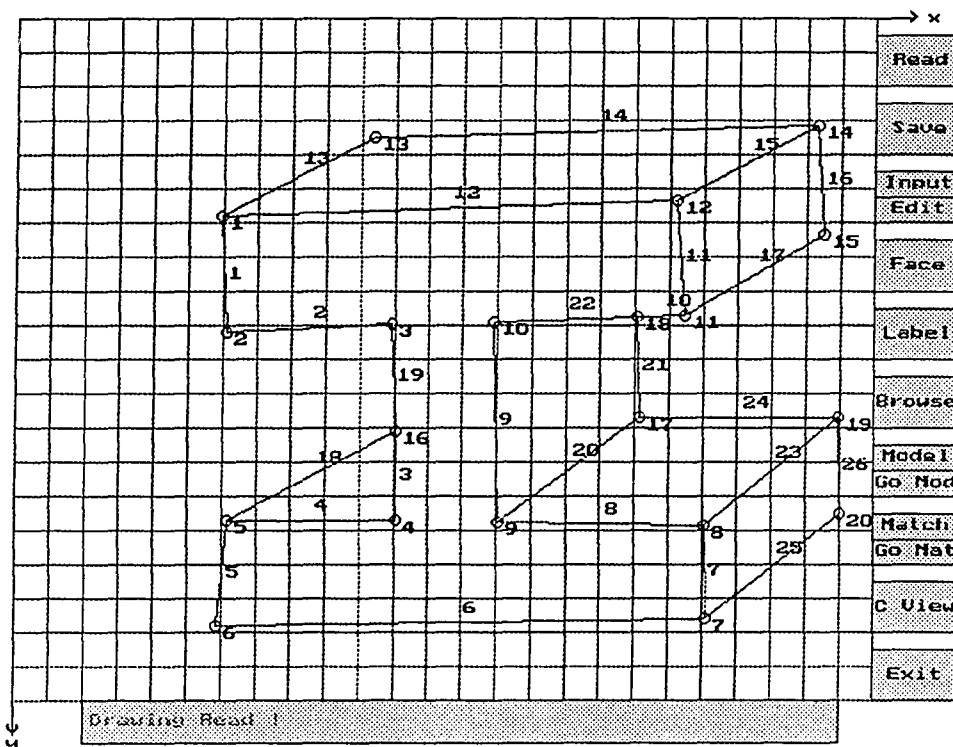
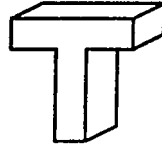


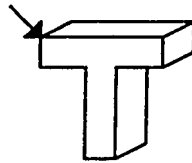
Figure A.2 - Reading obj7.2

Creating an object

Reading an object is easy. Creating a new one is a little more complicated. First you have to decide what to create. For this walk-through you want to create a 3-dimensional T shown below.



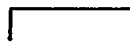
To do that, click on the *Input* button on top portion of the *Input/Edit* button. Now, you will have to draw this 3-dimensional T in the *Drawing Window*. First, you have to pick a point to start. Let us pick this point.



Click the left mouse button somewhere near the center of the *Drawing Window*. Now, move the mouse and there will be a dotted line to show where the line would be if you click on the left mouse button again. Now, position the mouse about an inch to the right of the starting point, and click on the left mouse button to draw this line.



3DOMMS has a built-in function that will snap two end points to connect the line if the two end points are close together. So click the left mouse button near the first point, move the mouse down about a quarter of an inch, and click the left mouse button again to create this drawing.



If the two endpoints are not close together, they will not snap together. If that happens, you will have to redraw the whole thing or *Edit* it. (Explained later) To redraw the drawing, click on the right mouse button to exit *Input* mode, click the left mouse button on the *Input* button, and draw the two lines again.

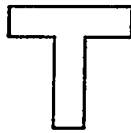
Drawing the third line the same way you drew the second line except that you start at the right end of the first line.



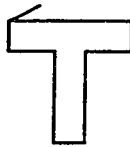
Now draw the line using the same method as before to create this.



Now, you should be able to finish drawing the T as shown below.

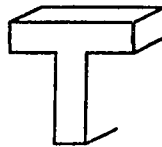


This is a T, but it is only 2-dimensional. Start at the upper left edge and draw a slanted line like this.



You can notice that the three end points snapped together to form a three-line intersection. This program will not allow a four-line intersection. A four-line will cause an error when the program is trying to *Face* or *Label* it. (Explained later)

Now draw five more lines to make the drawing look like this.



The next line that you have to draw is a different kind. It is a line in which you start at an endpoint and end in the middle of another line. If you start or end a line near the middle of another line, the second line will be broken into two small segments, and a T-junction will be formed where the second line is broken. Now try it by drawing the last line. What you have drawn should look something like **Figure A.3** on the next page.

3D OBJECT MODELING/MATCHING SYSTEM

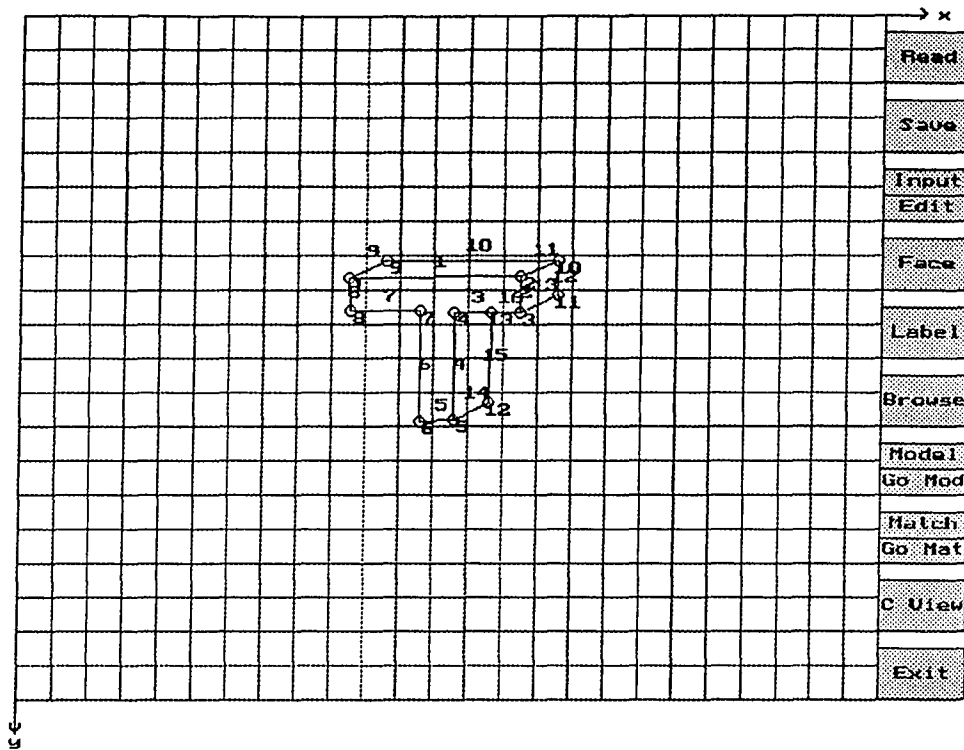


Figure A.3 - Inputting 3-dimensional T

The drawing is finished. As you can see each line is numbered and each edge is numbered. Click on the right mouse button to exit the *Input* mode.

Saving an object

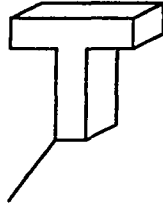
Now, you should save this object. Click on the *Save* button, and you will be asked for a filename to save this object as. Type *t* and press Enter. This 3-dimensional object will be saved as the file *t*.

Editing an object

Sometimes, the drawing may have errors, or maybe you just want to change the drawing in some way. To do that you will have to *Edit* it. Click on the *Edit* button on the bottom portion of the *Input/Edit* button.

The *Edit Sub Command Window* will appear. This menu contains these four options: *Insert*, *Delete*, *Pan*, and *Return*.

If you want to insert a line, click the left mouse button on the *Insert* button and start inserting lines just as if you were in the *Input* mode. Insert a line so that the drawing looks like this.



Click on the right mouse button to return to the *Edit Sub Command Window*. This does not look right. It is not a valid 3-dimensional object. So delete that line you just drew. Click the left mouse button on the *Delete* button. The *Delete* mode is different from the *Input* mode. In the *Delete* mode, you have to click on the line to select a line to be deleted. More than one line can be selected. Selected lines will have a dark red color. So, click the left mouse button on that extra line and that line will turn dark red. Now, click the right mouse button to delete that line and return to the *Edit Sub Command Window*.

Sometimes when your drawing is too big, you might run out of space because the program will not let you draw past the edges of the grid *Drawing Window*. You have to pan the object away from the edge to finish drawing what you started. Click the left mouse button on the *Pan* button. The *Pan* mode is different from both the *Delete* mode and the *Input* mode. In the *Pan* mode, you have to click on a node (intersection) to pick up the whole drawing. Then click somewhere else to place that node there, and the rest of the drawing will be placed accordingly.

So now click the left mouse button on any node you want. Click the left mouse button anyplace you want to move the whole object to. To leave the *Pan* mode, click on the right mouse button.

II. Canonical View Database Management -

The 3DOMMS simulation comes with a set of 12 objects. The current canonical view database contains the canonical views of these 12 objects. Each object has a different number of valid views. The valid views of these 12 objects are stored in the current model base. (See Section E on further details on managing these files.)

Browsing all the available canonical views

The *Browse* command is used to view all the canonical views that are currently in the CV database. Click on the *Browse* button and the *Browse Sub Command Menu* will appear. In the *Browse Sub Command Menu*, there are these three options: *PREVpg*, *NEXTpg*, and *Return*. The canonical views of the first 6 objects are shown above the *Browse Window* (as shown in Figure A.4 below).

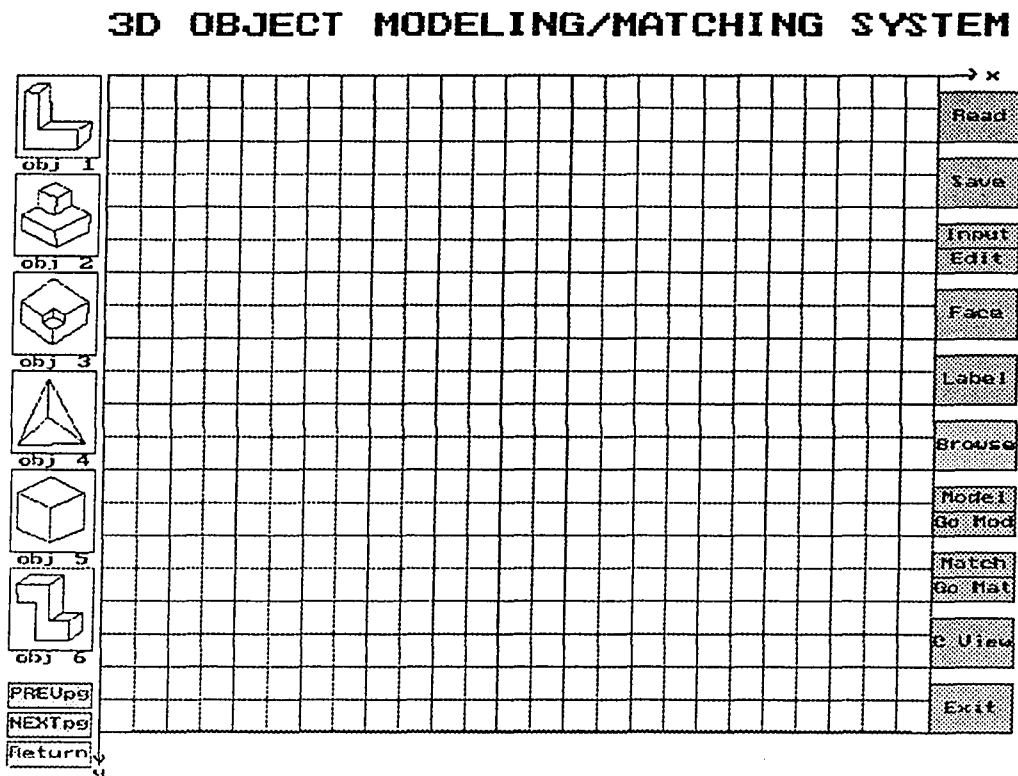


Figure A.4 - Browse

Click on *NEXTpg* to see the next 6 objects. Since there are currently only 12 objects, there are only 2 pages to browse. Click on *Return* to return to the *Main Command Window*.

To add more objects into the CV database, just *Insert* a new canonical view. To delete an object from the CV database, just *Delete* a canonical view. Now you will learn how to add the 3-dimensional T into the CV database.

Inserting a canonical view

If you do not have 3-dimensional T on the screen, then read the file *t*. Now, click on the *C View* command in the *Main Command Window*. The *C View Sub Command Window* should appear. This menu contains three options: *Insert*, *Delete*, and *Return*. Click on the *Insert* button. The program will ask for the object number. Since there's already object 1 through 12, 13 makes the most sense. Type **13** and press Enter.

Click on the *Return* button to return to the *Main Command Window*. Now, do *Browse* again and you will see that now there is 13 objects in the CV database. The 13th object is your 3-dimensional T as shown in **Figure A.5** on the next page.

Deleting a canonical view

Sometimes, you might want to delete an object from CV database. To do that, click on *C View* button and then click on the *Delete* button. When the program asks for the object number to be deleted, you can type whichever you want. (Note: If you accidentally deleted one of the original 12 objects and you want it back, see **Section E**.) Try deleting object 13 yourself. After deleting object 13, it will not appear in the *Browse Window*. Remember to *Insert* object 13 again if you deleted it just now.

3D OBJECT MODELING/MATCHING SYSTEM

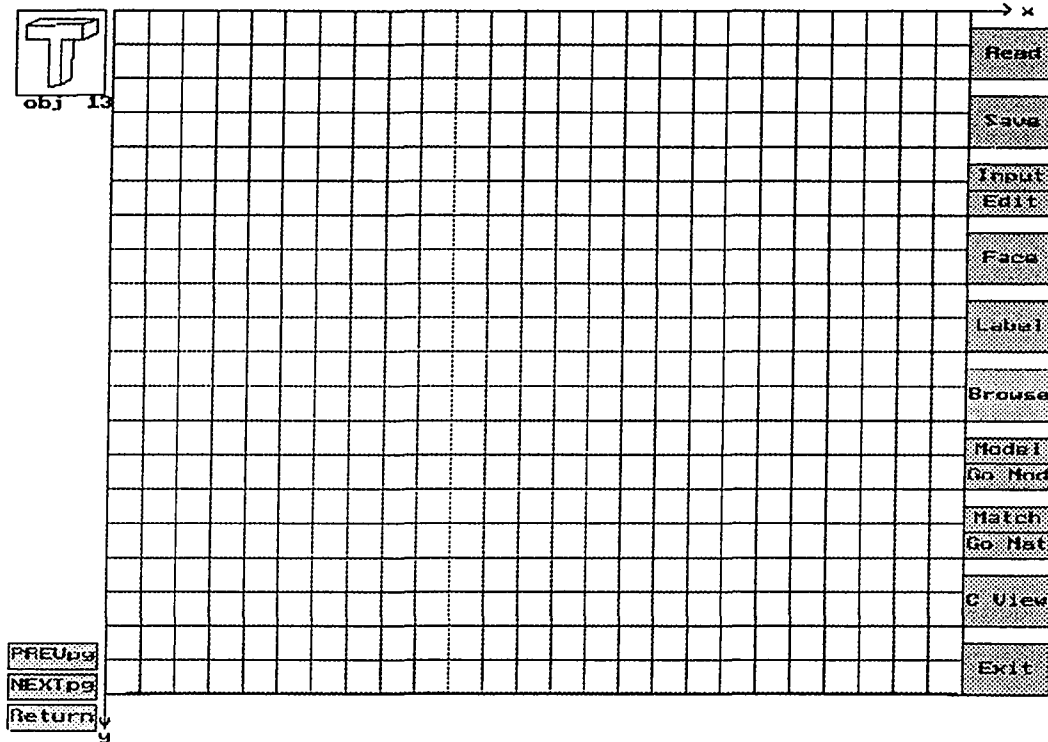


Figure A.5 - Browsing T

III. Object Modeling -

Now that you have inserted your own object, it is time to tell the program the different views that this object has. The first view should obviously be the T you just drew. Now *Read* file t.

Extracting faces from the object

Now that you have the first view of the 3-dimensional T on the screen, you should extract its faces. Click the left mouse button on the *Face* button. You should see something like Figure A.6 on the next page.

3D OBJECT MODELING/MATCHING SYSTEM

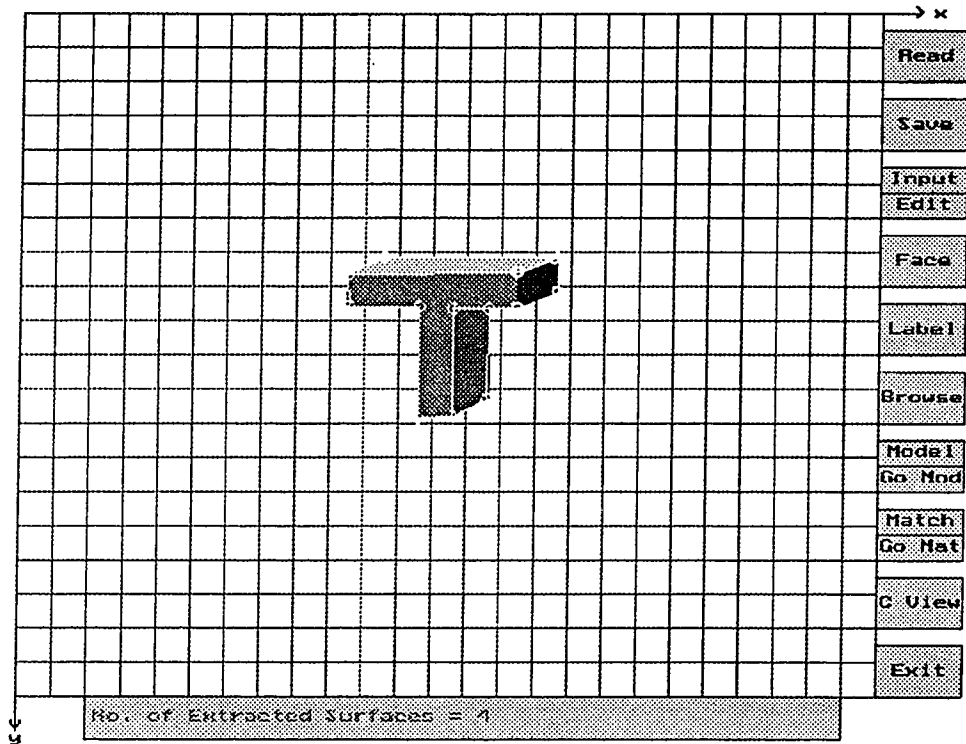


Figure A.6 - Extracting Faces of T

Labeling object

This view has four faces as you can see. Since there were no errors in this drawing, you can proceed and label this drawing. Click the left mouse button on the *Label* button. The screen should display something like **Figure A.7** on the next page.

3D OBJECT MODELING/MATCHING SYSTEM

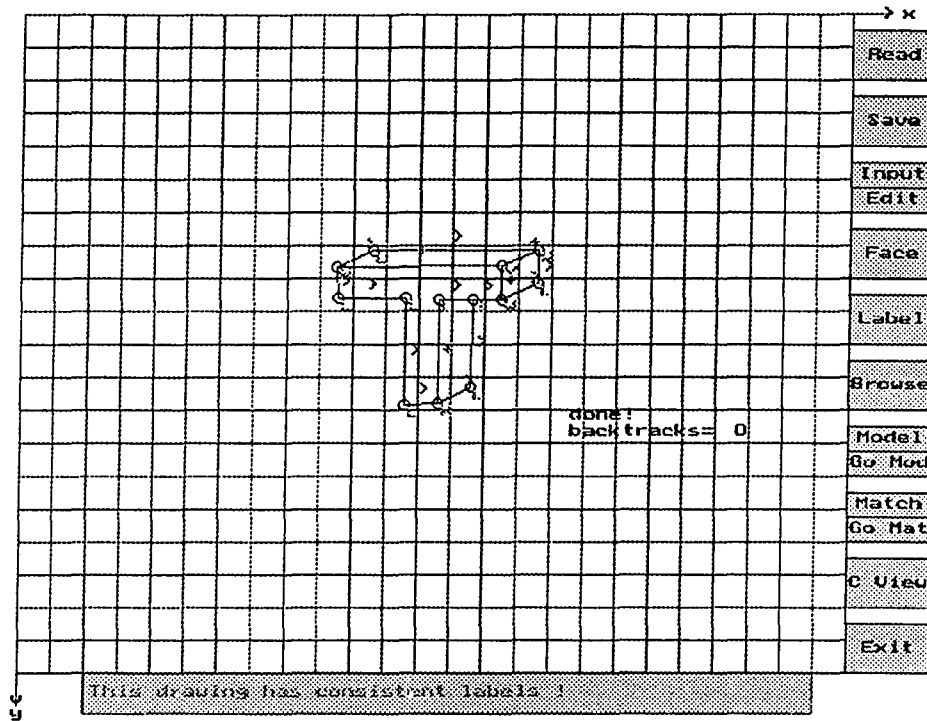


Figure A.7 - Labeling T

Each junction is labeled with one of these three labels: L, Y, or W.

L ~ This junction is an L-junction formed by two lines. (It looks like an L.)

Y ~ This junction is a Y-junction formed by three lines. (It looks like a Y.)

W ~ This junction is a W-junction formed by three lines. (It looks like a W.)

T ~ This junction is a T-junction formed by an end point breaking up a line.

(It looks like a T.)

Each line is labeled with one of these three labels: +, -, or >.

+ ~ This line is a convex line. (It is pointing outwards like a mountain.)

- ~ This line is a concave line. (It is pointing inwards like a valley.)

> ~ This line is an occlude line. (One of the face of this edge is occluded.)

The number of backtracks is displayed next to the labeled drawing.

Modeling object

If you did not successfully do the *Face* and the *Label* command for the 3-dimensional T you drew, then the drawing must have errors in it. Check your drawing and *Edit* it. Now, you can *Model* this object. Modeling an object just means that you are telling the program that this view is a valid view of an object in the CV database. Click the left mouse button on the *Model* button on the top portion of the *Model/Go Mod* button. The *Model Sub Command Window* will appear. This menu contains these three options: *Insert*, *Delete*, and *Return*. You want to *Insert* this view as a valid view for object 13 in the CV database. Click the left mouse button on the *Insert* button. The program will inform you that the largest object number is 13. In another words, there is no object number greater than 13. Since the view you have is a valid view for the obj13 (the 3-dimensional T), type **13** and press Enter. The program will inform you that this view has been added to model base.

Sometimes you might want to delete a view from a certain object. For example, if you added a view, that is not a valid view of object 13, to object 13. You will want to *Delete* that view. To *Delete* a view, just *Read* that file, Click on *Delete* in the *Model Sub Command Window*, and type the object number in which you want to delete that view from. (Note: Remember to *Insert* that view back into the model base so that 3DOMMS still knows that view as a valid view for that object.)

Go directly to model

Sometimes you might want to skip the process of doing *Face*, doing *Label*, and then doing *Model*. You can do all that in one step by clicking the left mouse button on the *Go Mod* button on the bottom portion of the *Model/Go Mod* button. This will get you directly to the *Model Sub Command Window* only if the drawing does not have any errors and the drawing has consistent labels.

IV. Object Matching -

Object matching is very similar to object modeling, except for that this time you tell the program to look for an object in the CV database that has the current view as a valid view. For example, the computer knows that the drawing in the file **obj9.2** is a valid view for object 9, if you match that drawing, the program will tell you that it object 9 is the only object in the CV database that contains **obj9.2**. *Read file obj9.2* (Figure A.8).

3D OBJECT MODELING/MATCHING SYSTEM

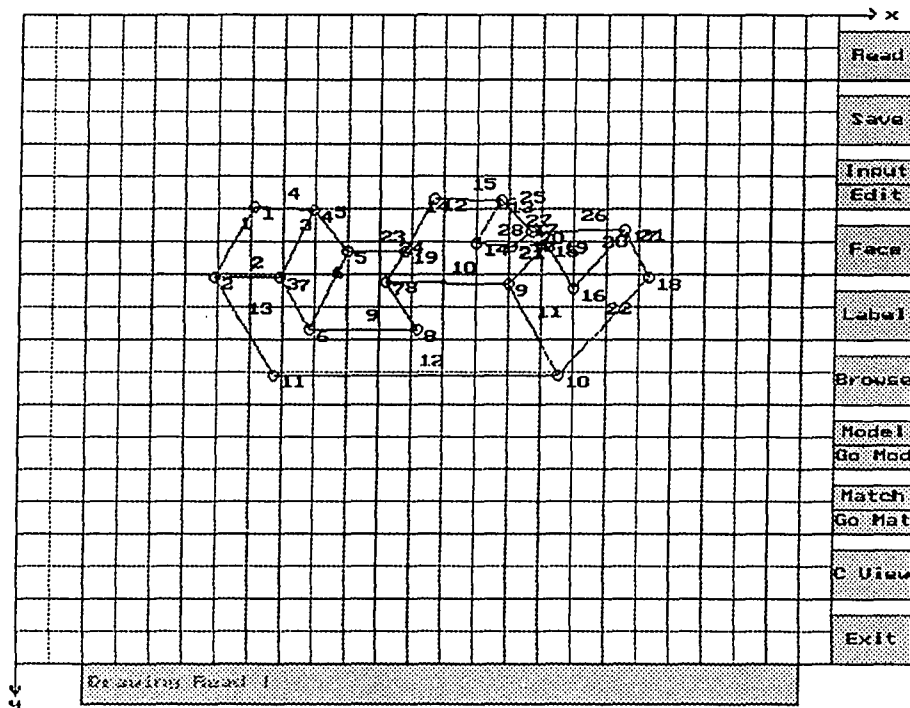


Figure A.8 - obj9.2

Extracting faces from the object

Extract the faces for this object as explained in **Object Modeling**.

Labeling object

Label this object as explained in **Object Modeling**

Matching object

Click the left mouse button on the *Match* button on the top portion of the *Match/Go Mat* button. The program will *Match* this object and it will display the canonical view of the object that contains this valid view as shown in **Figure A.9**.

3D OBJECT MODELING/MATCHING SYSTEM

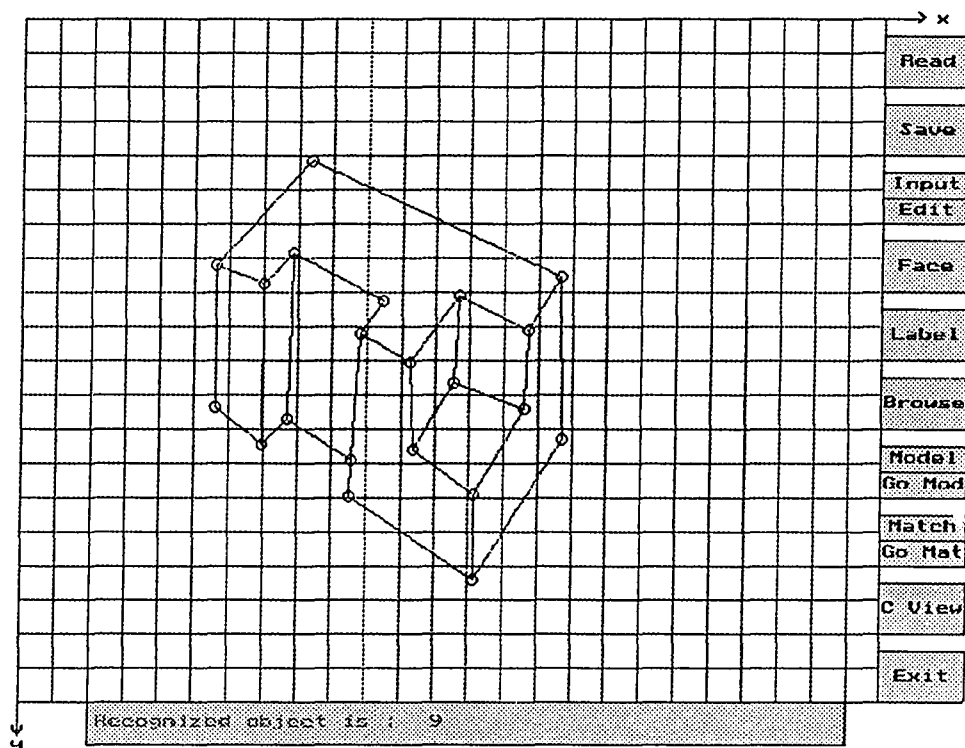


Figure A.9 - Canonical View for object 9

Go directly to match

Go Mat is just like *Go Mod*, except that it does *Face*, *Label*, and *Match* in one simple click.

Matching object that needs more than one view

Sometimes a drawing might match more than one object. The 3-dimensional T you drew is an example. If you try to *Match* that, you will find that there are 5 objects in the CV database that has the exact same signature as file *t*. This means that each of the 4 other objects in the CV database has each one view that has the exact number of faces, W-junctions, Y-junctions, T-junctions, + lines, and - lines, > lines !!! That seems like a highly unlikely thing, but it happens.

Read t. Click on the *Go Mat* button on the bottom portion of the *Match/Go Mat* button. The program will ask if there are more views of this object. The program will only ask this, if it is not sure that only one object in the CV database is the object that contains all the view you've given. Since there is only one view for object 13, type *n* and press Enter. The program now shows 5 objects from the CV database: object 9, 10, 11, 12, and 13. These 5 objects are all likely candidates. Until the program could eliminate them to one object, it will keep on asking for more input.

To show how the program can be sure of one object from a list of 3, use the files **obj1.1**, **obj1.2** and **obj1.4**. Objects 1 and 8 both contain a view that has the same signature as **obj1.1**. Objects 1, 2, and 11 all contain a view that has the same signature as **obj1.2**. Objects 1 and 2 both contain a view that has the same signature as **obj1.4**. In order for the program to recognize the object as object 1, you will have to enter as many of these views until the computer is fairly sure that it is object 1. If you only enter **obj1.1** and **obj1.2**, the program will say that objects 1,2, 8, and 11 are all likely candidates.

Now enter **obj1.1**, **obj1.2**, and **obj1.4**. Now, the program did not ask for another view, because it is sure that the it is object 1, and it shows that canonical view of object 1.

E. The Program

I. Files

Main Files

<i>m1.exe</i>	<i>read.exe</i>	<i>save.exe</i>
<i>input.exe</i>	<i>edit.exe</i>	<i>face.exe</i>
<i>label.exe</i>	<i>browse.exe</i>	<i>nreadpg.exe</i>
<i>preadpg.exe</i>	<i>match.exe</i>	<i>model.exe</i>
<i>cview.exe</i>	<i>cate2.dat</i>	<i>egavga.bgi</i>

Data Files

cv1 ... cv12
model.dat

Backup Data Files

cv_obj1 ... cv_obj12
model12.dat

Drawing Files

<i>obj1.1 ... obj1.5</i>	<i>obj2.1 ... obj2.16</i>	<i>obj3.1 ... obj3.6</i>
<i>obj4.1</i>	<i>obj5.1</i>	<i>obj6.1 ... obj6.4</i>
<i>obj7.1 ... obj7.5</i>	<i>obj8.1 ... obj8.27</i>	<i>obj9.1 ... obj9.25</i>
<i>obj10.1 ... obj10.17</i>	<i>obj11.1 ... obj11.34</i>	<i>obj12.1 ... obj12.6</i>

II. Description

Main Files

These files are the skeleton of the program. All of these files are needed in order for the program to run properly.

Data Files

These files contain the data for the current CV database and the current model base. The file *model.dat* contains the data for the model base. The files *cv1 ... cv12* contain the data for the current 12 objects that are in the CV database.

Currently, these 13 files contain the CV database and the model base for the 12 original objects. (Note: The data for these 12 objects could be cleared by deleting these 13 files) All the valid views of the objects in the CV database are stored in *model.dat*. When the user models a drawing, the changes are made to *model.dat* accordingly.

Each object in the CV database is stored individually the file *cv#*. (# is the object number) When the user inserts a new object into the CV database, a new *cv#* file will be created. When the user deletes an object from the CV database, a *cv#* will be deleted.

Backup Data Files

These 13 files contain the CV database and the model base for the 12 original objects. These files are just a copy of the *cv1 ... cv12* and *model.dat* files that came with the program.

If a *cv#* file is deleted, there are two ways in which this data could be replaced. The first way is to copy the file *cv_obj#* to file *cv#*. The second way is to *Read* the drawing file that contains the canonical view of the object # and then *Insert* this drawing as a canonical view into the CV database using the *Insert* option under the *C View* command. Here are the canonical view drawing files for the 12 original objects:

<i>obj1.1</i>	<i>obj2.1</i>	<i>obj3.1</i>	<i>obj4.1</i>
<i>obj5.1</i>	<i>obj6.1</i>	<i>obj7.4</i>	<i>obj8.1</i>
<i>obj9.17</i>	<i>obj10.8</i>	<i>obj11.1</i>	<i>obj12.3</i>

If the *model.dat* file is accidentally deleted or corrupted, there are two ways to replace that data. The first way is to *Read* all the drawings and *Model* all of them, which will take a long time. The second and easier way is to just copy the file *modell2.dat* to the file *model.dat*. (Note: *modell2.dat* only contains the model base for the 12 original objects. Any other data that the user added will be deleted if the file *model.dat* is replaced by the file *modell2.dat*. It is advised that backup copies of the file *model.dat* are frequently made.)

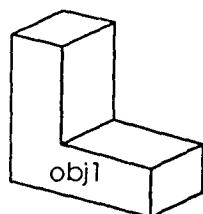
Drawing Files

This program comes with 147 drawing files that contain different valid views of the 12 original objects. The files are all named *obj#.\$* where # is the object number and \$ is the placing of the drawing.

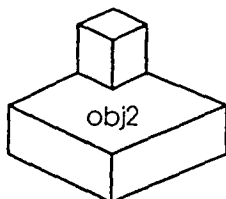
APPENDIX B

EXPERIMENTAL MODEL BASE

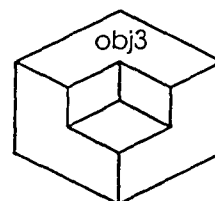
We have selected 12 model objects for the experiments. Some of them are obtained from [Huan93]. They are all trihedral objects, but are not constrained to be convex or right-angled trihedrons. The number of edges of the tested line-drawings are ranged from 6 (obj4) to 30 (obj8). The drawings shown in Figure B.1 are the canonical views of the corresponding objects. The line labels are also drawn in the appropriate positions. Some statistical information are listed below each object. (Note: the first row in vector form stands for the feature vector as defined in <Definition 6.5>; **S** is the signature of the drawing and is defined in <Definition 6.6>; **#E** is the number of edges of the drawing; **#B** is the number of backtracking during the labeling process; **# of Valid views** is the number of valid views stored in the current model database for the specified model object.)



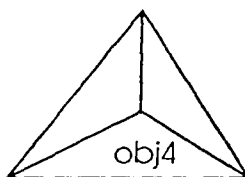
<5,6,1,8,3,5,0,3>
S=1231, #E=15, #B=0
of Valid Views=4



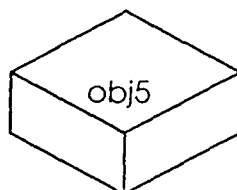
<6,6,2,10,4,6,0,3>
S=1435, #E=18, #B=0
of Valid Views=12



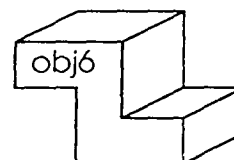
<6,9,3,0,4,6,0,3>
S=1595, #E=12, #B=0
of Valid Views=6



<3,3,0,3,1,3,0,0>
S=644, #E=6, #B=0
of Valid Views=1



<3,3,0,6,1,3,0,3>
S=695, #E=9, #B=0
of Valid Views=1



<5,6,1,10,3,5,0,5>
S=1265, #E=17, #B=0
of Valid Views=3

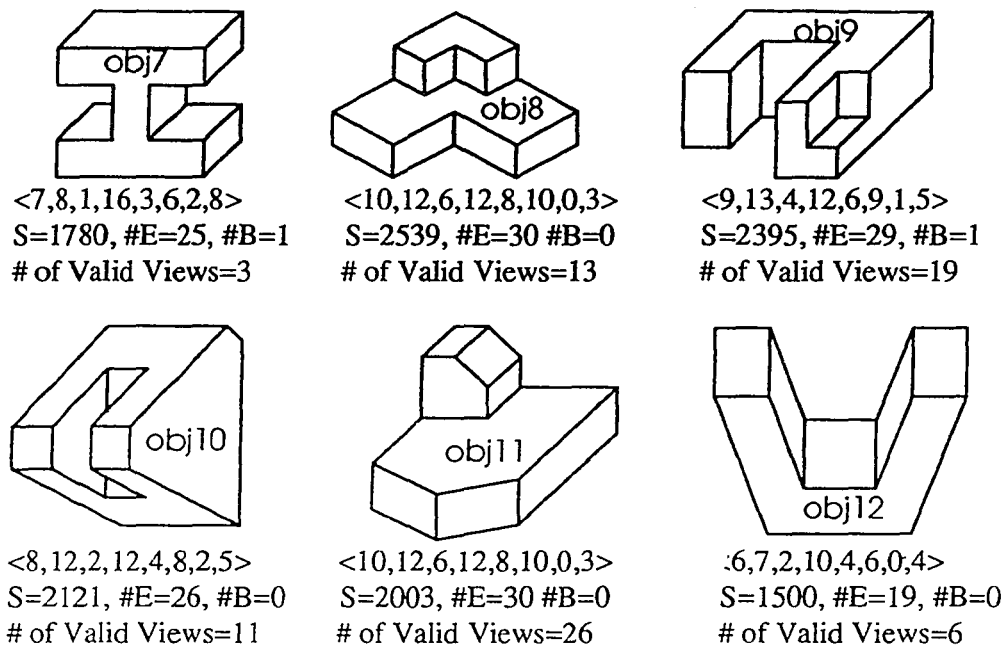


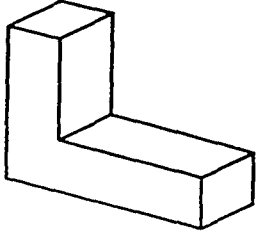
Figure B.1 - Experimental Model Base

The rest of drawings included in this appendix are all valid views for these 12 model objects. Their corresponding signature values and file names are also shown along with the drawings.

obj1 - 4 valid views

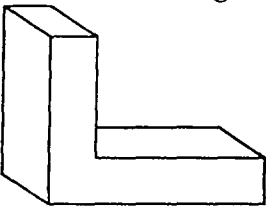
View #1 (cv) Signature 1231

File obj1.1



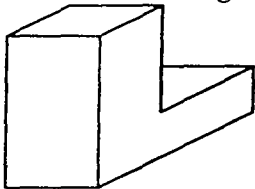
View #2 Signature 959

Files obj1.2 and obj1.3



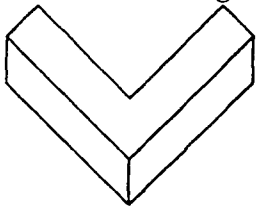
View #3 Signature 942

File obj1.4



View #4 Signature 729

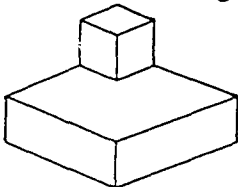
File obj1.5



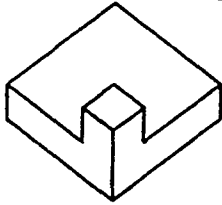
obj2 - 12 valid views

View #1 (cv) Signature 1435

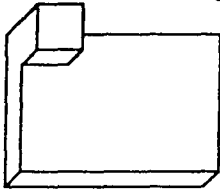
File obj2.1



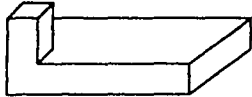
View #2 Signature 1026 File obj2.2



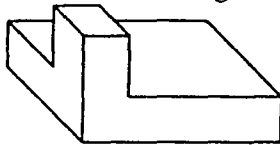
View #3 Signature 1259 Files obj2.3 and obj2.4



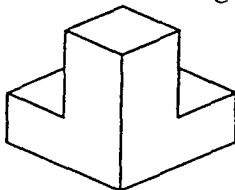
View #4 Signature 1242 File obj2.5



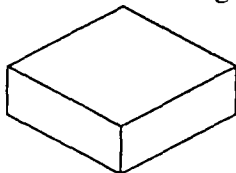
View #5 Signature 1206 Files obj2.6 and obj2.7



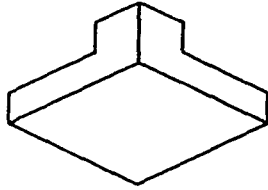
View #6 Signature 1189 File obj2.8



View #7 Signature 695 File obj2.9



View #8 Signature 763 File obj2.10



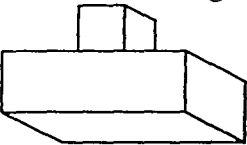
View #9 Signature 942 File obj2.11



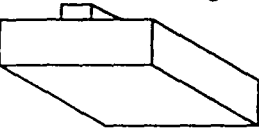
View #10 Signature 959 Files obj2.12 and obj2.13



View #11 Signature 1139 Files obj2.14 and obj2.15

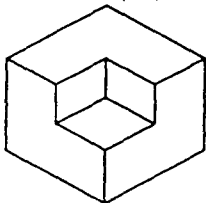


View #12 Signature 1122 File obj2.16

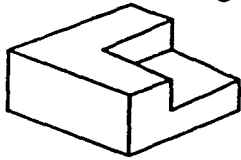


obj3 - 6 valid views

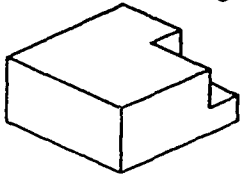
View #1 (cv) Signature 1595 File obj3.1



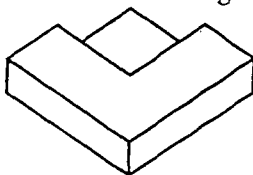
View #2 Signature 1211 File obj3.2



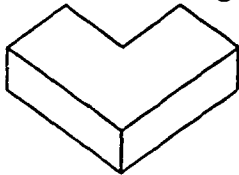
View #3 Signature 1189 File obj3.3



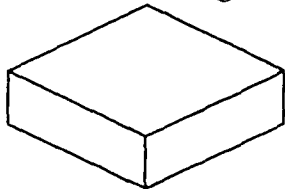
View #4 Signature 926 File obj3.4



View #5 Signature 729 File obj3.5

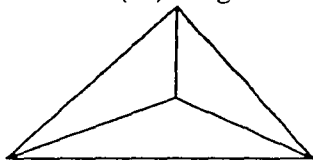


View #6 Signature 695 File obj3.6



obj4 - 1 valid view

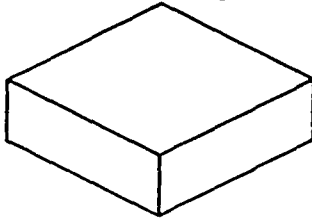
View #1 (cv) Signature 644 File obj4.1



obj5 - 1 valid view

View #1 (cv) Signature 695

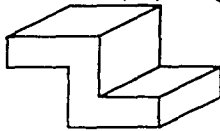
File obj5.1



obj6 - 3 valid views

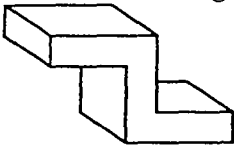
View #1 (cv) Signature 1265

File obj6.1



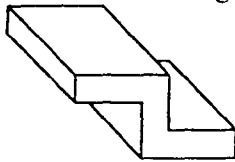
View #2 Signature 1223

Files obj6.2 and obj6.3



View #3 Signature 1206

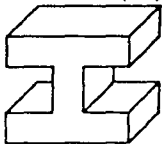
File obj6.4



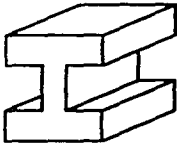
obj7 - 3 Valid Views

View #1 (cv) Signature 1793

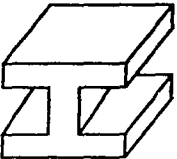
Files obj7.3 and obj7.4



View #2 Signature 1776 Files obj7.1 and obj7.2

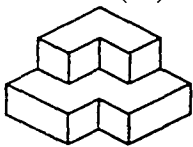


View #3 Signature 1770 File obj7.5

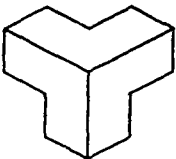


obj8 - 13 valid views

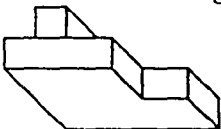
View #1 (cv) Signature 2539 File obj8.1



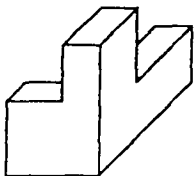
View #2 Signature 797 Files obj8.2 and obj8.25



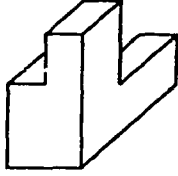
View #3 Signature 1658 Files obj8.3, obj8.4, and obj8.9



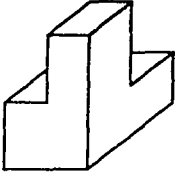
View #4 Signature 1223 Files obj8.5 and obj8.15



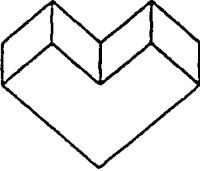
View #5 Signature 1206 Files obj8.6, obj8.8, and obj8.16



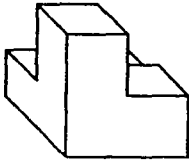
View #6 Signature 1189 File obj8.7



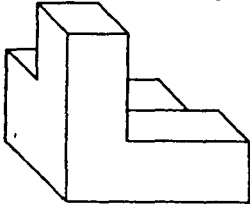
View #7 Signature 1231 File obj8.10



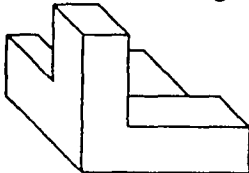
View #8 Signature 1386 File obj8.11



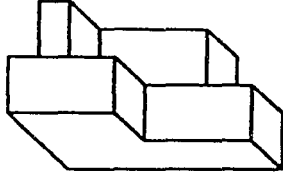
View #9 Signature 1403 Files obj8.12, obj8.13, and obj8.17



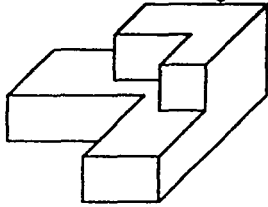
View #10 Signature 1420 Files obj8.14, obj8.18, obj8.21, obj8.22, and obj8.23



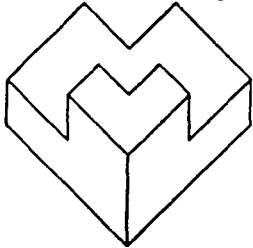
View #11 Signature 2107 File obj8.24



View #12 Signature 1787 File obj8.26

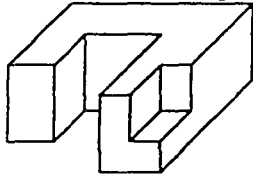


View #13 Signature 1094 File obj8.27

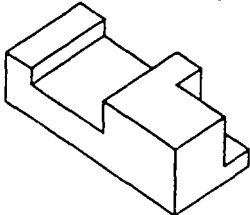


obj9 - 19 valid views

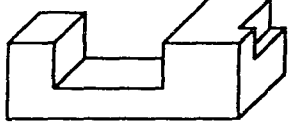
View #1 (cv) Signature 2395 File obj9.17



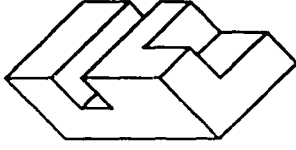
View #2 Signature 1989 Files obj9.1 and obj9.13



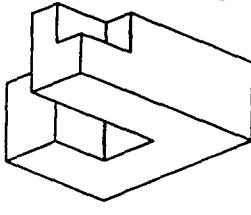
View #3 Signature 2011 Files obj9.2 and obj9.16



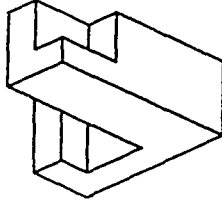
View #4 Signature 2005 File obj9.3



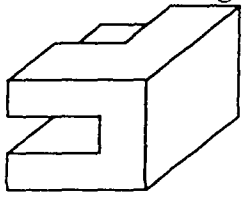
View #5 Signature 2003 File obj9.4



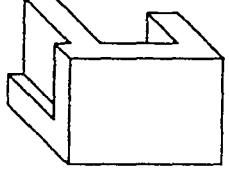
View #6 Signature 1986 File obj9.5



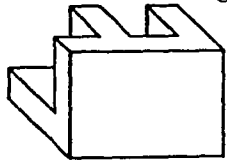
View #7 Signature 1207 Files obj9.6 and obj9.9



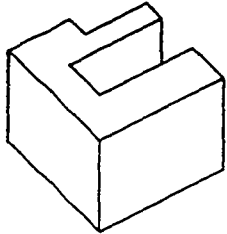
View #8 Signature 1492 File obj9.7



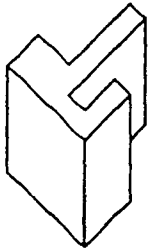
View #9 Signature 1470 File obj9.8



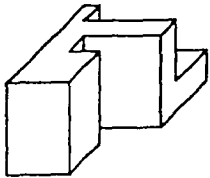
View #10 Signature 1010 Files obj9.10 and obj9.21



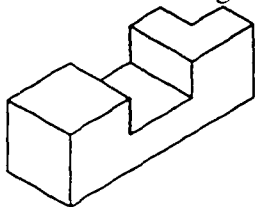
View #11 Signature 1027 File obj9.11



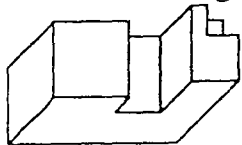
View #12 Signature 1983 Files obj9.12 and obj9.25



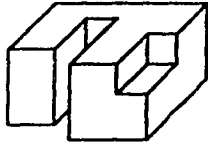
View #13 Signature 1529 File obj9.14



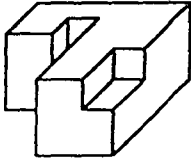
View #14 Signature 1726 File obj9.15



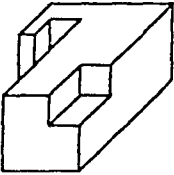
View #15 Signature 2389 File obj9.18



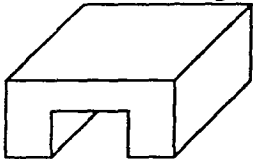
View #16 Signature 2387 File obj9.19



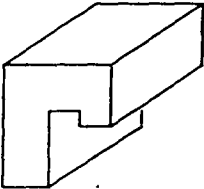
View #17 Signature 2370 File obj9.20



View #18 Signature 976 Files obj9.22 and obj9.23

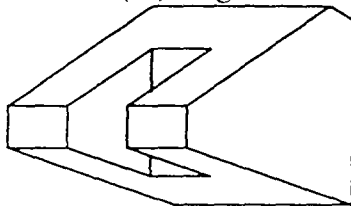


View #19 Signature 993 File obj9.24

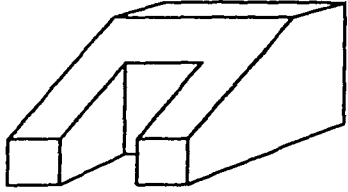


obj10 - 11 valid views

View #1 (cv) Signature 2121 File obj10.8

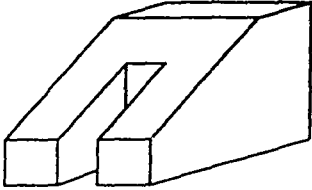


View #2 Signature 1779



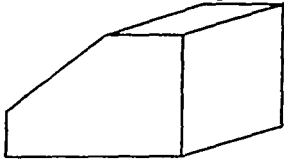
File obj10.1

View #3 Signature 1773



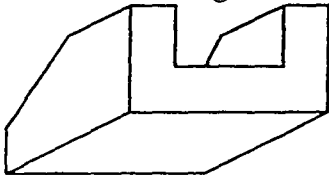
Files obj10.2 and obj10.13

View #4 Signature 712



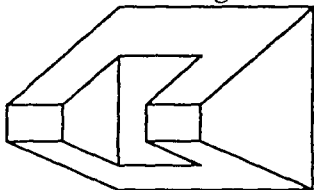
File obj10.3

View #5 Signature 1010



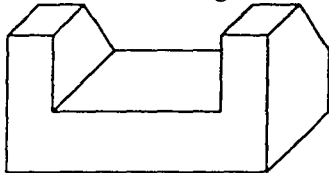
Files obj10.4 and obj10.15

View #6 Signature 1925



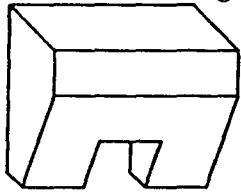
File obj10.5

View #7 Signature 1512



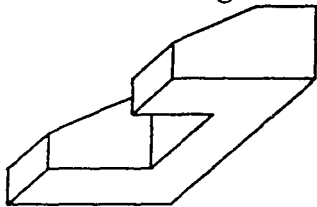
Files obj10.6 and obj10.11

View #8 Signature 1277



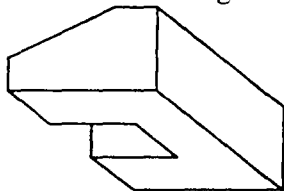
File obj10.7

View #9 Signature 1506



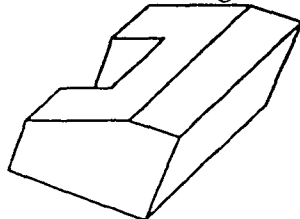
Files obj10.9 and obj10.10

View #10 Signature 993



Files obj10.12 and obj10.14

View #11 Signature 1260

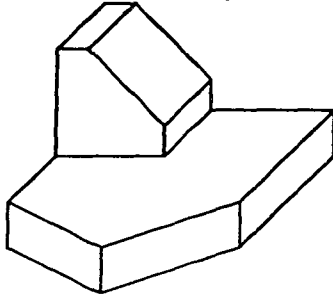


Files obj10.16 and obj10.17

obj11 - 26 valid views

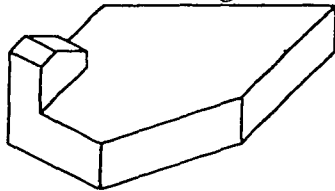
View #1 (cv) Signature 2003

File obj11.1



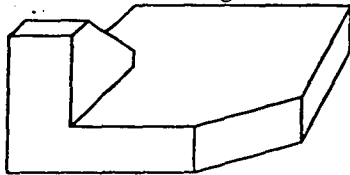
View #2 Signature 1827

Files obj11.2 and obj11.4



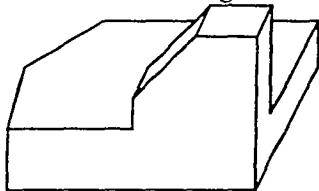
View #3 Signature 1560

File obj11.3



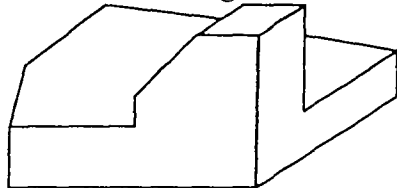
View #4 Signature 1507

Files obj11.5 and obj11.11

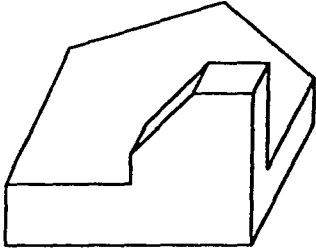


View #5 Signature 1240

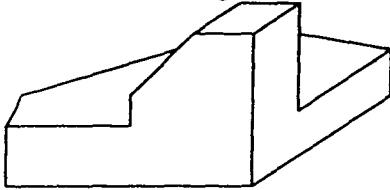
Files obj11.6 and obj11.9



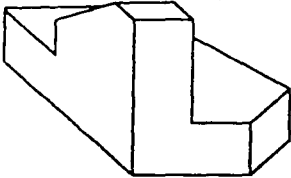
View #6 Signature 1327 File obj11.7



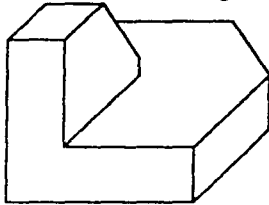
View #7 Signature 1223 File obj11.8



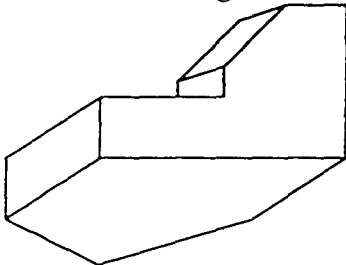
View #8 Signature 1490 File obj11.10



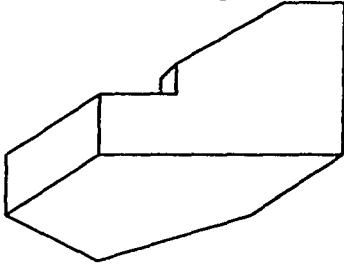
View #9 Signature 1276 File obj11.12



View #10 Signature 1260 Files obj11.13 and obj11.22

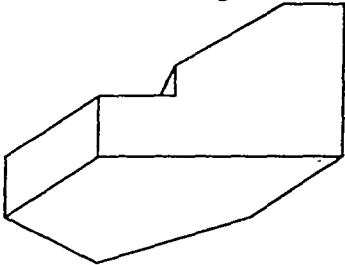


View #11 Signature 993



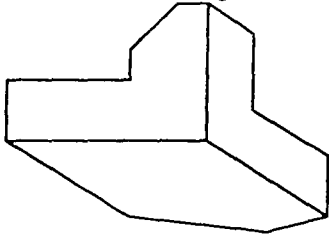
Files obj11.14 and obj11.18

View #12 Signature 976



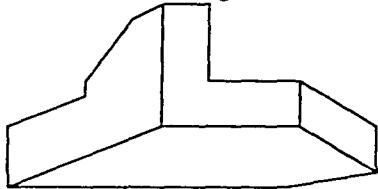
Files obj11.15 and obj11.19

View #13 Signature 797



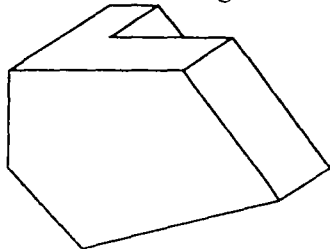
File obj11.16

View #14 Signature 1064



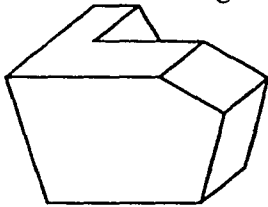
File obj11.17

View #15 Signature 959

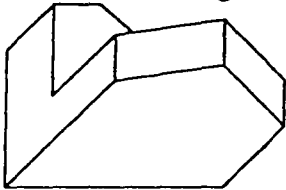


File obj11.20

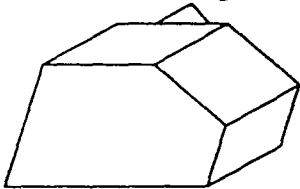
View #16 Signature 1226 File obj11.21



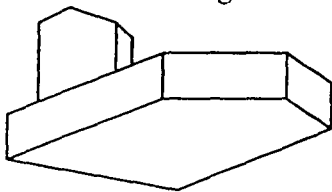
View #17 Signature 1243 Files obj11.23 and obj11.35



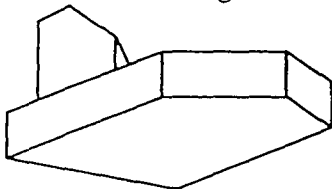
View #18 Signature 1176 File obj11.24



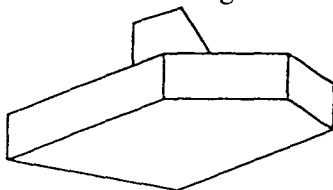
View #19 Signature 1440 Files obj11.25 and obj11.29



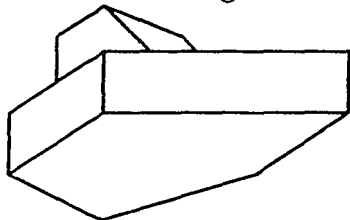
View #20 Signature 1423 Files obj11.26 and obj11.27



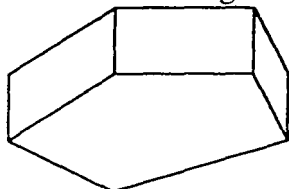
View #21 Signature 1193 File obj11.28



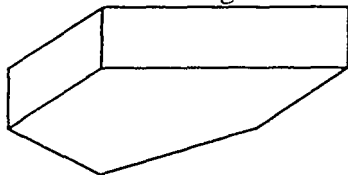
View #22 Signature 1156 File obj11.30



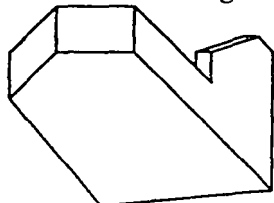
View #23 Signature 979 File obj11.31



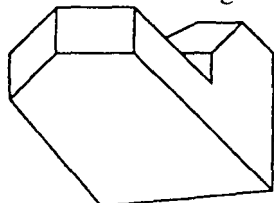
View #24 Signature 712 File obj11.32



View #25 Signature 1527 File obj11.33



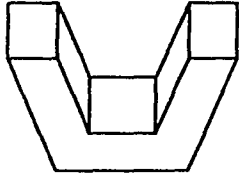
View #26 Signature 1525 File obj11.34



obj12 - 6 valid views

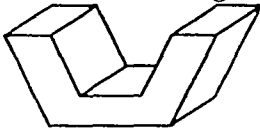
View #1 (cv) Signature 1500

File obj12.3



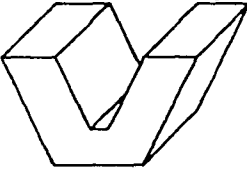
View #2 Signature 1495

File obj12.1



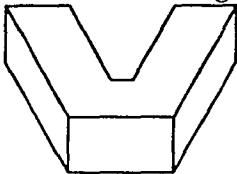
View #3 Signature 1489

File obj12.2



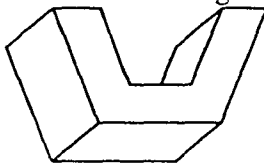
View #4 Signature 1030

File obj12.4



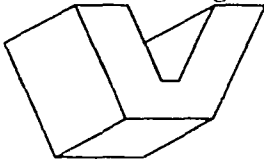
View #5 Signature 993

File obj12.5



View #6 Signature 976

File obj12.6



BIBLIOGRAPHY

- [Bena90] J. Ben-arie, "Probabilistic models of observed features and aspects with application to weighted aspect graph," *Pattern Recognition*, Vol.11, pp.421-427, 1990.
- [Berg87] F. Bergholm, "Edge Focusing," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.PAMI-9, No.6, pp.726-741, 1987.
- [Berz73] A. T. Berztiss, "A Backtrack Procedure for Isomorphism of Directed Graphs," *J. of Association for Computing Machinery*, Vol.20, No.3, pp.365-377, 1973.
- [BesJ85] P. J. Besl and R. C. Jain, "Three-Dimensional Object Recognition," *Computing Surveys*, Vol.17, No.1, pp.75-145, 1985.
- [BhaH87] B. Bhanu and C. C. Ho, "CAGD-Based 3D Object Representations for Computer Vision," *IEEE Computer*, Vol.20, No.8, pp.19-36, 1987.
- [Bley84] H. Bley, "Segmentation and Preprocessing of Electrical Schematics Using Picture Graph," *Computer Vision, Graphics, and Image Processing*, Vol.28, pp.271-288, 1984.
- [BowD90] K. W. Bowyer and C. R. Dyer, "Aspect Graphs: An Introduction and Survey of Recent Results," *International Journal of Imaging Systems and Technology*, Vol.2,, pp.315-328 1990.
- [Bunk82] H. Bunke, "Attributed Programmed Graph Grammars and Their Application to Schematic Diagram Interpretation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.PAMI-4, No.6, pp.574-582, 1982.

- [Cann86] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.PAMI-8, No.6, pp.679-698, 1986.
- [CFMP84] U. Cugini, G. Ferri, P. Mussio, and M. Protti, "Pattern-directed Restoration and Vectorization of Digitized Engineering Drawings," *Comput. Graphics*, Vol.8, pp.337-350, 1984.
- [ChaH92] I. C. Chang and C. L. Huang, "Aspect Graph Generation for Non-Convex Polyhedra from Perspective Projection View," *Pattern Recognition*, Vol.25, No.10, pp.1075-1096, 1992.
- [Chak79] I. Chakravarty, "A Generalized Line and Junction Labeling Scheme with Applications to Scene Analysis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.1, pp.202-205, 1979.
- [Chen91] T. Cheng, "Reasoning 3-D Relationships from 2-D Drawings by Planning," *Proceeding of IEEE OCEAN'91*, pp.1088-1093, 1991.
- [Chen92] T. Cheng, "Segmenting Symbols from Engineering Drawings by Image Abstraction and Blurring," *Proceeding of The First International Conference on Automation, Robotics, and Computer Vision (ICARCV'92)*, pp.cv-2.7.1-2.7.5, 1992.
- [CheY94a] T. Cheng and D. Y. Y. Yun, "Canonical Modeling and Multi-view Indexing for 3-D Object Recognition," accepted for presentation at *The Third International Conference on Automation, Robotics, and Computer Vision (ICARCV'94)*, Singapore, 1994.
- [CheY94b] T. Cheng and D. Y. Y. Yun, "Drawing Labeling by a Cascaded Planning Model - CCRP," accepted for presentation at *The Second Singapore International Conference on Intelligent Systems (SPICIS'94)*, Singapore, 1994.

- [ChiD86] R. T. Chin and C. R. Dyer, "Model-Based Recognition in Robot Vision," *Computing Surveys*, Vol. 18, No.1, pp.67-108, 1986.
- [ChKi92] S. B. Cho and J. H. Kim, "A Two-Stage Classification Scheme with Backpropagation Neural Network Classifiers," *Pattern Recognition Letters*, 13, pp.309-313, 1992.
- [CKLY93] T. Cheng, J. Khan, H. Liu, and D. Y. Y. Yun, "A Symbol Recognition System," *Proceeding of The Second International Conference on Document Analysis and Recognition*, pp.918-921, 1993.
- [CleJ91] D. T. Clemens and D. W. Jacobs, "Model Group Indexing for Recognition," *ICPR'91*, pp.4-8, 1991.
- [Clem81] T. P. Clement, "The Extraction of Line-structured Data from Engineering Drawings," *Pattern Recognition*, Vol.14, pp.43-52, 1981.
- [Clow71] M. B. Clowes, "On Seeing Things," *Artificial Intelligence*, Vol.2, No.1, pp.79-116, 1971.
- [Coop93] M. C. Cooper, "Interpretation of Line Drawing of Complex Objects," *Image and Vision Computing*, Vol.11, No.2, pp.82-90, 1993.
- [CorG70] D. G. Corneil and C. C. Gotlieb, "An Efficient Algorithm for Graph Isomorphism," *J. of Association for Computing Machinery*, Vol.17, No.1, pp.51-64, 1970.
- [DiPR92] S. J. Dickinson, A. P. Pentland, and A. Rosenfeld, "3D Shape Recovery Using Distributed Aspect Matching," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-14, No.2, pp.174-198, 1992.
- [DuBM77] B. A. Dudani, K. J. Breeding and R. B. McGhee, "Aircraft Identification by Moment Invariants," *IEEE Trans. on Computer*, Vol.C-26, pp.39-46, 1977.
- [EBDC92] D. W. Eggert, K. W. Bowyer, C. R. Dyer, H. I. Christensen, D. B. Goldgof, "The Scale Space Aspect Graph," *ICV'92*, pp.335-340, 1992.

- [EdOS83] H. Edelsbrunner, J. O'Rourke and R. Seidel, "Constructing Arrangements of Lines and Hyperplanes with Applications," *Proc. of 23rd Symp. on Foundations of Computer Science*, 1983.
- [EggB90] D. Eggert and K. Bowyer, "Computing the Orthographic Projection Aspect Graph of Solids of Revolution," *Pattern Recognition Letters*, Vol.11, pp.751-763, 1990.
- [FanC91] K. C. Fan and C. Y. Chang, "Surface Extraction from Line Drawings of a Polyhedron," *Pattern Recognition Letters*, 12, pp.627-633, 1991.
- [Faug92] O. Faugeras, J. Mundy, N. Ahuja, C. Dyer, A. Pentland, R. Jain, K. Ikeuchi, K. Bowyer, "Panel theme: Why aspect graphs are not (yet) practical for computer vision," *CVGIP: Image Understanding*, 55, pp.212-218, 1992.
- [FlyJ91] P. J. Flynn and A. K. Jain, "CAD-Based Computer Vision: From CAD Models to Relational Graphs," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-13, No.2, pp.114-132, 1991.
- [Fly92] P. J. Flynn, "Saliencies and Symmetries: Toward 3D Object Recognition from Large Model Databases," *ICV'9*, pp.322-3272, 1992.
- [FuKE84] M. Furuta, N. Kase and S. Emori, "Segmentation and Recognition of Symbols for Handwritten Piping and Instrument Diagram," *ICPR'84*, pp.626-629, 1984.
- [Fuka84] Y. Fukada, "A Primary Algorithm for the Understanding of Logic Circuit Diagrams," *Pattern Recognition*, Vol.17, pp.125-134, 1984.
- [FuYC93] C. W. Fu, J. C. Yen and S Chang, "Calculation of Moment Invariants via Hadamard Transform," *Pattern Recognition*, Vol.26, pp.287-294, 1993.
- [GigM88] Z. Gigus and J. Malik, "Computing the Aspect Graph for Line Drawings of Polyhedral Objects," *IEEE Int. Conf. on Robotics and Automation*, pp.1560-1566, 1988.

- [GiMA83] B. Gil, A. Mitiche and J. K. Aggarwal, "Experiments in Combining Intensity and Range Edg Maps," *Computer Vision, Graphics, and Image Processing*, Vol.21, pp.395-411, 1983.
- [GrSS85] F. C. A. Groen, A. C. Sanderson and F. Schlag, "Symbol Recognition in Electrical Diagrams Using Probabilistic Graph Matching," *Pattern Recognition Letters*, Vol.3, pp.343-350, 1985.
- [GuHu85] W. K. Gu and T. S. Huang, "Connected Line Drawing Extraction from a Perspective View of a Polyhedron," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.7, pp.422-430, 1985.
- [GuYH87] W. K. Gu, J. Y. Yang and T. S. Huang, "Matching Perspective Views of Polyhedron Using Circuits," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.7, No.3, pp.320-326, 1987.
- [Guzm68] A. Guzman, "Decomposition of a Visual Scene into Three-dimensional bodies," *AFIPS Proc. Fall Joint Comp. Conf.*, 33, pp.291-304, 1968.
- [HaII85] H. Harada, Y. Itoh and M. Ishii, "Recognition of Freehand Drawings in Chemical Plant Engineering," *Proceedings of IEEE Workshop on Computer Architecture for Pattern Recognition and Image Database Management*, pp.146-153, 1985.
- [Hama93] A. H. Hamada, "A New System for the Analysis of Schematic Diagrams," *Proceeding of the Second International Conference on Document Analysis and Recognition*, pp.369-372, 1993.
- [HarS79] R. M. Haralick and L. G. Shapiro, "The Consistent Labeling Problem: Part I," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No.2, pp.173-184, 1979.
- [HorS79] E. Horowitz and S Sahni, *Fundamental of Computer Algorithms*, Pitman, Lodon.

- [HoYH85] W. Ho, D. Y. Y. Yun, and Y. H. Hu, "Planning Strategies for Switchbox Routing," *Proceeding of the International Conference on Computer Design*, pp.463-467, 1985.
- [Huan93] C. L. Huang, "Pictorial Drawing Generation for Polyhedral Object Recognition Using Intensity Images," *Pattern Recognition Letters*, pp.121-131, 1993.
- [Hu61] M. K. Hu, "Pattern recognition by moments invariants," *Proc. IRE*, 49, pp.1428, 1961.
- [Huff71] D. A. Huffman, "Impossible Objects as Nonsense Sentences," in B. Meltzer and D. Nichie Eds. *Machine Intelligence*, Vol.6, Edinburgh Univ. Press, pp.295-324, 1971.
- [JiaB93] X. Y. Jiang and H. Bunke, "An Optimal Algorithm for Extracting the Regions of a Plane Graph," *Pattern Recognition Letters*, 14, pp.553-558, 1993.
- [Jose89] S. H. Joseph, "Processing of Engineering Line Drawings for Automatic Input to CAD," *Pattern Recognition*, Vol.22, No.1, pp.1-11, 1989.
- [Kana80] T. Kanade, "A theory of Origami World," *Artificial Intelligence*, Vol.13, pp.279-311, 1980.
- [KBES90] R. Kasturi, S. T. Bow, W. El-Masri, J. Shah, J. R. Gattiker and U. B. Mokate, "A System for Interpretation of Line Drawings," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.PAMI-12, No.10, pp.978-992, 1990.
- [KenF87] J. R. Kender and D. G. Freudenstein, "What is a Degenerate View," *IJCAI'87*, pp.801-804, 1987.
- [KenY89] N. Keng and D. Y. Y. Yun, "A Planning/Scheduling Methodology for the Constrained Resource Problem," *IJCAI'89*, pp.20-25, 1989.

- [Keng89] N. Keng, "A General Planning/Scheduling Paradigm for the Constrained Resource Planning Problem," Ph.D dissertation, Computer Science Department, Southern Methodist University, 1989.
- [KeYR88] N. P. Keng, D. Y. Y. Yun, and M. Rossi, "Interactive-sensitive Planning System for Job-shop Scheduling," *Expert Systems and Intelligent Manufacturing*, M. Oliff (ed.), North-Holland, pp.57-69, 1988.
- [KhLu88] A. Khotanzad and J. H. Lu, "Distortion Invariant Character Recognition by a Multi-Layer Perceptron and Back-Propagation Learning," *IJCNN'88*, pp.I-625-632, 1988.
- [Kiro90] L. M. Kirousis, "Effectively Labeling Planar Projections of Polyhedra," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.12, pp.123-130, 1990.
- [KiSK93] S. H. Kim, J. W. Suh and J. H. Kim, "Recognition of Logic Diagrams by Identifying Loops and Rectilinear Polylines, " *Proceeding of the Second International Conference on Document Analysis and Recognition*, pp.349-352, 1993
- [KodM86] K. L. Kodanpani and E. J. McGrath, "A Wirelist Compare Program for Verifying VLSI Layouts," *IEEE Design and Test of Computers*, Vol.3, pp.46-51, 1986.
- [Kolm63] A. N. Kolmogorov, "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition," *Dokl. Akad. Nauk SSSR*, 144, 676-681; *Amer. Math. Soc. Transl.*, 28, pp.55-59, 1963.
- [KorD87] M. R. Korn and C. H. Dyer, "3-D Multiview Object Representations for Model-Based Object Recognition," *Pattern Recognition*, Vol.20, No.1, pp.91-103, 1987.

- [LamB93] D. Lamb and A. Bandopadhyay, "Shape from Line Drawings: Beyond Huffman-Clowes Labeling," *Pattern Recognition Letters*, Vol.14, pp.213-219, 1993.
- [Lee92] S. W. Lee, "Recognizing Hand-drawn Electrical Circuit Symbols with Attributed Graph Matching," *Structured Document Image Analysis*, Eds. by H. S. Baird, H. Bunke, and K. Yamamoto, Springer-Verlag, pp.340-358, 1992.
- [Li93] B. C. Li, "A New Computation of Geometric Moments," *Pattern Recognition*, Vol.26, pp.109-113, 1993.
- [LieC90] W. N. Lie and Y. C. Chen, "Robust Line-Drawing Extraction for Polyhedra Using Weighted Polarized Hough Transform," *Pattern Recognition*, Vol.23, pp.261-274, 1990.
- [LiuS90] H. C. Liu and M. D. Srinath, "Corner Detection from Chain-code," *Pattern Recognition*, Vol.23, No.1/2, pp.51-68, 1990.
- [LiYD88] H. H. Liu, T. Y. Young, and A. Das, "A Multilevel Parallel Processing Approach to Scene Labeling Problems," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.10, No.4, pp.586-590, 1988.
- [LSMS85] X. Lin, S. Shimotsuji, M. Minoh and T. Sakai, "Efficient Diagram Understanding with Characteristic Pattern Detection," *Computer Vision, Graphics, and Image Processing*, Vol.30, pp.84-106, 1985.
- [Mack73] A. K. Mackworth, "Interpreting Pictures of Polyhedral Scenes," *Artificial Intelligence*, Vol.8, pp.121-137, 1973.
- [Mali87] J. Malik, "Interpreting Line Drawings of Curved Objects," *International Journal of Computer Vision*, Vol.1, 1987, pp.73-103.

- [MavB93] J. Maver and R. Bajcsy, "Occlusions as a Guide for Planning the Next View," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.15, No.5, pp.417-433, 1993.
- [Nalw88] V. S. Nalwa, "Line-Drawing Interpretation: Straight Lines and Conic Sections," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.10, No.4, pp.514-529, 1988.
- [OKMT88] A. Okazaki, T. Kondo, K. Mori, S. Tsunekawa, and E. Kawamoto, "An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.10, No.3, pp.331-341, 1988.
- [Pav186] T. Pavlidis, "A Vectoriser and Feature Extractor for Document Recognition," *Comput. Vision Graphics Image Process.* Vol.35, pp.111-127, 1986.
- [Regi91] T. Regier, "Line Labeling and Junction Labeling: A Coupled System for Image Interpretation," *IJCAI'91*, pp.1305-1310, 1991.
- [Rose87] A. Rosenfeld, "Recognizing Unexpected Objects: A Proposed Approach," *Int. J. of Pattern Recognition and Artificial Intelligence*, Vol.1, No.1, pp.71-84, 1987.
- [RuHW86] D. E. Rumelhart, G. E. Hilton and R. J. Williams, "Learning Representations by Backpropagating Errors," *Nature*, Vol.323, No.9, pp.533-536, 1986.
- [RPAK88] A. P. Reeves, R. J. Prokop, S. E. Andrews and F. P. Kuhl, "Three-Dimensional Shape Analysis Using Moments and Fourier Descriptors," *IEEE Trans. Pattern Analysis Mach. Intell.*, PAMI-10, pp.937-943, 1988.
- [SalY91] G. J. Salem and T. Y. Young, "A Neural Network Approach to the Labeling of Line Drawings," *IEEE Trans. on Computers*, Vol.40, No.12, pp.1419-1424, 1991.

- [Sham78] M. I. Shamos, "Computational Geometry," Ph.D. dissertation, Computer Science Department, Yale University, New Haven, CT, 1978.
- [ShLY89] Z. Shih, R. C. T. Lee and S. N. Yang, "A Systolic Algorithm for Extracting Regions from a Planar Graph," *Comp. Vision Graphics Image Process.* 47, pp227-242, 1989.
- [SaSB90] M. Sallam, J. Stewman and K. Bowyer, "Computing the Visual Potential of an Articulated Assembly of Parts," *ICV'90*, pp.636-643, 1990.
- [SteB88] J. Stewman and K. Bowyer, "Creating the Perspective Projection Aspect Graph of Polyhedral Objects," *ICV'88*, pp.494-500, 1988.
- [SteB90] J. H. Stewman and K. Bowyer, "Direct Construction of the Perspective Projection Aspect Graph of Convex Polyhedra," *Computer Vision, Graphics, and Image Processing*, Vol.51, pp.20-37, 1990.
- [Sugi86] K. Sugihara, *Machine Interpretation of Line Drawing*, The MIT Press, Cambridge, Massachusetts, London, England, pp.12-39, 1986.
- [ToHL84] J. T. Tou, C. L. Huang and W. H. Li, "Design of a Knowledge-Based System for Understanding Electronic Circuit Diagrams," *Proc. First Conf. on Artificial Intelligent Applications*, Denver, USA, pp.652-661, 1984.
- [TsaL90] C. K. Tsao and W. C. Lin, "Constrait Propagation Neural Networks for Huffman-Clowes Scene Labeling," *Proceeding of IEEE Second International Conference on Tools for A.I.*, pp.262-268, 1990.
- [Ullm76] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *J. of Association for Computing Machinery*, Vol.23, No.1, pp.31-42, 1976.
- [Umey88] S. Umeyama, "An Eigendecomposition Approach to Weighted Graph Matching Problems," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol.10, No.5, pp.695-703, 1988.

- [Walt72] D. Waltz, "Generating Semantic Descriptions from Drawings of Scenes with Shadows," *Technical Report AI-TR-271*, MIT, 1972.
- [WanF90] R. Wang and H. Freeman, "The Use of Characteristic-View Classes for 3D Object Recognition," in *Machine Vision for Three-Dimensional Scenes*, Academic Press, Inc., pp.109-161, 1990.
- [Watt88] N. A. Watts, "Calculating the Principal Views of a Polyhedron," *ICPR'88*, pp.316-322, 1988.
- [WoKI91] K. C. Wong, J. Kittler and J. Illingworth, "Heuristically Guided Polygon Finding," *British Machine Vision Conference*, pp.400-407, 1991.
- [WonY85] A. K. C. Wong and M. You, "Entropy and Distance of Random Graphs with Application to Structural Pattern Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No.5, pp.599-609, 1985.
- [Wong92] Wong, E. K., "Model Matching in Robot Vision by Subgraph Isomorphism," *Pattern Recognition*, Vol.25, No.3, pp.287-303, 1992.
- [YouW84] M You and A. K. C. Wong, "An Algorithm for Graph Optimal Isomorphism," *ICPR'84*, pp.316-319, 1984.
- [YuQC92] D. Y. Y. Yun, B. G. Qiu and Z. L. Chen, "Solving Traveling Salesman Problems by Planning," *Proceeding of the International Conference on Intelligent Information Processing and Systems*, (invited paper), Beijing, China, Oct. 1992.