THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical Engineering
College of Engineering

SEL Technical Report No. 51

OPTIMUM IMPLEMENTATION OF TOPOLOGICAL STRUCTURES FOR INTERACTIVE COMPUTER DISPLAYS

by

James H. Jackson

under the direction of
Professor Keki B. Irani

January 1971

under contract with:

ROME AIR DEVELOPMENT CENTER
Research and Technology Division
Griffiss Air Force Base, New York
Contract No. F30602-69-C-0214
and
ADVANCED RESEARCH PROJECTS AGENCY
Department of Defense
Washington, D.C.
Contract No. DA-49-083 OSA-3050

PREFACE

This research could not have been completed without the support of the Advanced Research Projects Agency and Rome Air Development Center, or without the assistance of many people. The author is indebted to the Advanced Research Projects Agency, contract number DA-49-083 OSA-3050, for the opportunity to implement computer graphics programs during preliminary phases of this research, and for financial support at that time. He is also indebted to Rome Air Development Center, contract number F30602-69-C-0214, for financial support during the latter part of the research.

The author is most grateful for the assistance of his doctoral committee. He is particularly indebted to Professor Irani, for his detailed reading of the manuscript and many helpful suggestions. The author's association with Professor Irani during preliminary related research is also greatly appreciated. Professor Westervelt has been a source of encouragement and guidance during the early phases of this research, which has proved to be most helpful. The reviews of the work by Professor Galler and Professor Herzog are also sincerely appreciated.

The final manuscript was typed by Miss Joyce Doneth, Mrs. Carolyn Heile, and the author's wife, Carole. Earlier versions

of the manuscript were typed by Miss Sharon Bauerle and Mrs. Joanne Aichler.

Finally, the author is indebted to his wife for her continued encouragement and support throughout this period of research.

TABLE OF CONTENTS

		Page
PREFACE		ix
LIST OF TABLES		xi
LIST OF FIGURES		xii
LIST OF APPENDICES		xv
NOTATION		xvi
Chapter		
1	INTRODUCTION	1
	1.1 Brief History of Topological Structures and Display Equipment	1
	1.2 Types of Implementation	6
	1.3 Types of Structure	7
	1.4 Objectives	8
2	THE TOPOLOGICAL STRUCTURE	9
	2.1 Topology	15
	2.1.1 Definition of Topology	17
	2.1.2 Dependent Representation of Entities	20
	2.1.2.1 Structural Equivalence	23
	2.1.2.2 Entities as Transformed Subpictures	30
	2.1.2.2.1 Representation of Coordinate Transformations	32
	2.1.2.2.2 Generation of Entities from Subpictures	35

Chapter Page

2.1.2.3 Classification of Entities 44
2.1.3 Representation of Topology 49
2.2 Coordinate Transformations 56
2.2.1 A Structure Which Enforces No Constraints 59
2.2.2 A Structure Which Enforces Concatenation Constraints 61
2.2.2.1 Ordering of Entities 61
2.2.2.2 Concatenation of Subpictures 63
2.2.2.2.1 Coordinate Systems for Concatenation 67
2.2.2.2.2 Transformations Performed by Drawing a Subpicture 68
2.2.2.2.3 Concatenations as Coordinate Transformations 72
2.2.2.3 Values of F Determined by Concatenation 78
2.2.2.4 Description of the Structure 79
2.2.2.5 Examples of the Structure 80
2.2.3 Applicability of Structures 86
2.3 Conclusion 89

Chapter		Page
3	COST	92
	3.1 Storage Requirement	93
	3.1.1 Overhead Storage	95
	3.1.2 Storage Per Element of B	99
	3.1.3 Storage Per Element of E/D	107
	3.1.4 Examples	111
	3.2 Computer Time Required to Generate a Picture	117
	3.2.1 An Algorithm for Generating the Picture	117
	3.2.2 Time Required to Perform Algorithm 3.1	125
	3.2.2.1 Initialization	125
	3.2.2.2 Identifying Entities from the Structure	126
	3.2.2.3 Displaying Entities in E_0	137
	3.2.2.4 Restoring from the Push-Down Stack	139
	3.2.2.5 Summary	141
	3.3 Average Response Time	142
	3.3.1 Definition of Basic Operations	144
	3.3.2 Time Required to Perform Basic Operations	148
	3.3.2.1 Time Required to Modify the Structure	152
	3.3.2.1.1 Creating and Destroying Elements of E/D	152

Chapter Page

3.3.2.1.2 Creating Elements of B 153
3.3.2.1.3 Destroying Elements of B 159
3.3.2.1.4 Modifying Elements of B in the Structure $\{B, E/D, \mu, \gamma\}$ 163
3.3.2.2 Time Required to Interrogate the Structure 164
3.3.2.2.1 Identifying P(D[e]) for a Given Class D[e] 164
3.3.2.2.2 Identifying Values of μ' 170
3.3.2.2.3 Identifying the Last Element of B Needed to Form a Subpicture 171
3.3.2.2.4 Retrieving $G(D[e])^{-1}$ from $D[e] \epsilon E_0/D_0$ 172
3.3.2.2.5 Identifying Values of μ 173
3.3.2.2.6 Identifying the First Component of an Element of B 174

Chapter Page

3.3.2.2.7 Identifying the Second Component of an Element of B 175
3.3.2.2.8 Retrieving Values of F' and γ 177
3.4 Conclusion 178

4 DESIGN DECISIONS 179
4.1 Definition of a Set of Design Decisions 179
4.2 Effect of Design Decisions on Storage 186
4.2.1 An Algorithm for Computing Sizes of Storage Blocks 187
4.2.2 Storage of Items Common to All Blocks 195
4.2.3 Storage of Items Which Describe an Element of B 197
4.2.4 Storage of Items Which Describe an Element of E/D 199
4.3 Effect of Design Decisions on Picture Generation Time 202
4.3.1 Time During Which the Computer is Devoted to Picture Generation 202
4.3.2 Time Which Elapses While the Picture is Generated 210
4.4 Effect of Design Decisions on Response Time 213
4.4.1 Operations Which Modify the Structure 214

Chapter		Page
	4.4.2 Operations Which Interrogate the Structure	218
	4.5 Constraints	223
	4.6 Conclusion	226
5	APPLICATIONS	228
	5.1 Methods of Implementation	229
	5.1.1 Storage Formats	230
	5.1.2 Elementary Instruction Sequences	231
	5.2 Time Required to Perform Algorithms 3.2 and 3.5	238
	5.2.1 Time Required to Compute $G(D[e])^{-1}$	239
	5.2.2 Time Required to Perform Algorithm 3.5	244
	5.3 Optimum Implementations for Specific Applications	246
	5.3.1 A Text Editing Program	246
	5.3.1.1 Estimation of Parameters	247
	5.3.1.2 Results of Analysis	260
	5.3.2 A Graph Theory Program	267
	5.3.2.1 Estimation of Parameters	269
	5.3.2.2 Results of Analysis	283
	5.4 Conclusion	289
6	CONCLUSIONS	291
	6.1 General Design Principles	291
	6.2 Limitations	293
	6.3 Strengths and Weaknesses	293
BIBLIOGRAPHY		351

LIST OF TABLES

		Page
2.1	Paths Which Represent Entities in Figure 2.10	55
4.1	Design Decisions	180
5.1	Storage Requirements for DEC 339	232
5.2	Elementary DEC 339 Instruction Sequences	234
5.3	Structure Size Parameters for Text Editor	250
5.4	Storage Parameters for Text Editor	251
5.5	Time Parameters for Text Editor	253
5.6	Values of the Function $\bar{n}$ for Text Editor	261
5.7	Structure Size Parameters for Graph Theory Program	271
5.8	Storage Parameters for Graph Theory Program	272
5.9	Time Parameters for Graph Theory Program	274
5.10	Values of the Function $\bar{n}$ for Graph Theory Program	278
A.1	Device Codes	305
A.2	Text Statements	309
A.3	Text Statements for Graph Theory Program	319
B.1	Notation for Parameters	336

LIST OF FIGURES

		Page
1.1	Interactive Computer Display Program as Described by W. R. Sutherland	2
2.1	A Picture and a Representation of Its Topology	11
2.2	Examples of Pictures for Which the Topology of Each Picture Facilitates Response to Control Language Inputs	13
2.3	Graphs of the Relations C and A for Picture in Figure 2.1	21
2.4	An Electrical Circuit Diagram and the Representation of Its Connectivity by the Relation A	22
2.5	Examples of the Subsets $E_0, E_1, E_2, \ldots, E_m$	25
2.6	An Example of Two Equivalent Structures	31
2.7	Some Simple Coordinate Transformation Matrices	36
2.8	Subpictures Used to Generate Picture in Figure 2.1	37
2.9	Coordinate Systems for Entities in Figure 2.3	41
2.10	Graphical Representation of B	53
2.11	An Ambiguous Topological Structure	58
2.12	Examples of Pictures for Which a Concatenation Facility is Useful	64
2.13	Shaw's Binary Concatenation Operators	66

		Page
2.14	Example Transformation Matrices G(D[e])	69
2.15	Concatenation of Two Subpictures	73
2.16	Other Concatenations of Subpictures in Figure 2.15	74
2.17	Applications of the Concatenation Facility	81
2.18	A Topological Structure for an Editor Program	83
2.19	Concatenations of Text Characters	85
2.20	Isolation of Coordinate Transformations in the Structure $\{B, E/D, \mu, \gamma\}$	88
2.21	Alternative Representations of Concatenation	90
3.1	A Storage Allocation Scheme	96
3.2	Data Blocks for a List Structure Representation of $\{B, E/D, \mu, \gamma\}$	112
3.3	Representation of Two Angles by a List Structure	113
3.4	Modification of the List Structure Blocks in Figure 3.2	116
3.5	A Relation A With Values of ψ Indicated	129
3.6	Part of a Structure $\{B, E/D, \mu, \gamma\}$	134
3.7	Example of Picture Distortion Due to Lack of Synchronization	155
4.1	Packing of Items Into Storage Blocks	191
5.1	Topological Structures for Text Editing	248

		Page
5.2	Estimation of T_{tf} and T_{tg} for Text Editor	258
5.3	Text Editor Cost for $n_\ell = 20$	264
5.4	Text Editor Cost for $n_c = 35$	265
5.5	Topological Structures for Graph Theory	268
5.6	Graph Theory Program Cost for $n_e = 100$ and $\ell = 100$	285
5.7	Graph Theory Program Cost for $n_v = 80$ and $\ell = 100$	286
5.8	Graph Theory Program Cost for $n_e/n_v = 5/4$ and $\ell = 100$	287
A.1	Block Formats for Elements of E/D	297
A.2	Block Formats for Elements $\rho(e,e') \epsilon B$	298
A.3	Representation of State Diagrams	303
A.4	Example Tracking Pattern Characters	306
A.5	State Diagram for Graph Theory Program	318
B.1	Flow Chart for the SOLVE Command	332
B.2	PAR Subroutine for Text Editor	334
B.3	Listing of Parameters for DEC 339	344

LIST OF APPENDICES

Appendix		Page
A	AN IMPLEMENTATION OF THE STRUCTURE $\{B, E/D, \mu, \gamma\}$	295
	A.1 Implementation of the Structure	295
	A.2 The Display Language	301
	A.2.1 The State Diagram	302
	A.2.2 Specification of Operations Associated with a State	307
	A.2.3 An Example of an Application Program	316
	A.3 Operation of the Program	323
	A.4 Suggested Improvements	327
B	PROGRAMS USED FOR ANALYSIS	329
	B.1 Enumeration	329
	B.2 Printing	346
	B.3 Application of Constraints	347
	B.4 Plotting	349

NOTATION

The following list summarizes the symbols which occur frequently in the text.

Symbols used to describe a topological structure:

Symbol	Meaning	Page
β	Function which maps subpictures into entities	35
γ	Function which maps elements of B into sequences of coordinate transformation matrices	75
δ, δ'	Entity ordering functions	62
ρ	Function which maps elements of A into elements of B	54
λ	Entire picture entity	18
μ	Function which generally maps each element of a set $\rho(\{e\} \times S(e))$ into the previous element in this set	79
μ'	Function which generally maps each element of a set $\rho(\{e\} \times S(e))$ into the next element in this set	101
ψ	Function which maps each entity into the number of times that it is displayed while the picture is generated once	127

Symbol	Meaning	Page
A	Intransitive topology relation	19
B	Set of instances	49
C	Transitive topology Relation	17
D	Relation which describes which entities may be represented as transformations of a common subpicture	44
D_0	Relation which describes which primitive entities are transformations of a common subpicture	45
E	Set of entities	17
E_0	Set of primitive entities	23
F	Function which maps elements of A into coordinate transformation matrices	42
F'	Function which maps elements of B into coordinate transformation matrices	59
G	Function which maps elements of E/D into coordinate transformation matrices	68
H	Function which maps entities into coordinate transformation matrices	38

Symbol	Meaning	Page
P	Function which maps each element D[e] ϵ E/D into the set of elements of B whose second component is D[e]	102
S'	Function which maps each entity into the smallest set of other entities of which it is composed	127
T	Relation which describes which entities have equivalent structures	23

Parameters to be estimated:

σ	Ratio of elapsed memory access time for picture generation to actual memory access time	212
$f_f(\ell_i), f_g(\ell_i)$	Frequencies of Operation ℓ_i	214
G_{bf}, G_{bg}	Times required for the computer to identify the second component of an element of B during the picture generation process	206, 207
G_{cf}, G_{cg}	Times required for the computer to convert subpictures for entities in E_0 into display processor format	208

Symbol	Meaning	Page
G_{hf}, G_{mf}	Times required for the computer to effectively multiply matrices while generating a picture from the structure $\{B,E/D,F'\}$	204, 205
G_{hf}', G_{mf}'	Times during which the computer is suspended while the display processor effectively multiplies matrices while generating a picture from the structure $\{B,E/D,F'\}$	204, 205
G_{hg}, G_{mg}	Times required for the computer to effectively multiply matrices while generating a picture from the structure $\{B,E/D,\mu,\gamma\}$ when references to second components of elements of B are stored	205, 206
G_{hg}', G_{mg}'	Times during which the computer is suspended while the display processor effectively multiplies matrices while generating a picture from the structure $\{B,E/D,\mu,\gamma\}$	205, 206
G_{hg}'', G_{mg}''	Times required for the computer to effectively multiply matrices while generating a picture from the structure $\{B,E/D,\mu,\gamma\}$ when references to second components of elements of B are not stored	205, 206

Symbol	Meaning	Page
G_s	Time required for the computer to save a reference to an element of B on a push-down stack and to restore this reference	203
G_s'	Time during which the computer is suspended while the display processor saves a reference to an element of B on a push-down stack and restores this reference	203
G_s''	Time required for the computer to save a value of H on a push-down stack and to restore this value	204
G_s'''	Time during which the computer is suspended while the display processor saves a value of H on a push-down stack and restores this value	204
G_{tf}, G_{tg}	Times during which the computer is suspended while entities in E_0 are transferred to the display processor	209
G_{tf}', G_{tg}'	Times during which the computer is suspended while subpictures in $\beta(E_0)$ are transferred to the display processor	209, 210

Symbols	Meaning	Page
q_{bf}, q_{bg}	Cardinalities of B	184
q_{df}, q_{dg}	Cardinalities of E/D	184
q_{d0f}, q_{d0g}	Cardinalities of E_0/D_0	184
q_{sf}, q_{sg}	$\sum_{e \in E - \{\lambda\}} \psi(e)$	184
q_{s0f}, q_{s0g}	$\sum_{e \in E_0} \psi(e)$	184
r_c	Time required to copy the contents of one location into another	153
$\bar{r}_d$	Average time required to destroy a storage block	153
$\bar{r}_g$	Average time required to create a storage block	153
R_γ, R_γ'	Average times required to determine the form of a value of γ	222
R_f, R_f'	Average times required to retrieve a value of F' from the structure $\{B, E/D, F'\}$	222

Symbol	Meaning	Page
R_g, R_g'	Average times required to retrieve a value of G for a class in E_0/D_0 from the structure $\{B, E/D, \mu, \gamma\}$	221, 222
R_j	Average time required to store a display processor jump command	215
R_p	Average time required to store a pointer	215
R_{sf}, R_{sg}	Average times required to perform Algorithm 3.5 when the computer performs Algorithm 3.1 and references to second components of elements of B are stored	219, 220
R_{sf}', R_{sg}'	Average times required to perform Algorithm 3.5 when the computer performs Algorithm 3.1 and references to second components of elements of B are not stored	220
R_{sf}'', R_{sg}''	Average times required to perform Algorithm 3.5 when the display processor performs Algorithm 3.1	220

Symbol	Meaning	Page
s_0	Overhead storage associated with creating and destroying a storage block	104
$S_{\beta f}, S_{\beta g}$	Average storage occupied by a subpicture for a class in E_0/D_0 when the computer forms entities in E_0	200
$S_{\beta f}', S_{\beta g}'$	Average storage occupied by a subpicture for a class in E_0/D_0 when the display processor forms entities in E_0	200
S_γ	Average storage required to represent a value of γ when the computer performs Algorithm 3.1	199
S_γ'	Average storage required to represent a value of γ when the display processor performs Algorithm 3.1	199
S_a	Storage per element $D[e] \epsilon E/D-E_0/D_0$ which is occupied by information which causes the display processor to compute $G(D[e])^{-1}$	201

Symbol	Meaning	Page
S_f	Average storage per element of B required to represent a value of F' when the computer performs Algorithm 3.1	198
S_f'	Average storage per element of B required to represent a value of F' when the display processor performs Algorithm 3.1	198
S_g	Average storage per element $D[e] \in E_0/D_0$ required to represent $G(D[e])^{-1}$ when the computer performs Algorithm 3.1	201
S_g'	Average storage per element $D[e] \in E_0/D_0$ required to represent $G(D[e])^{-1}$ when the display processor performs Algorithm 3.1	201
S_h	Storage occupied by a display processor halt command	202
S_j	Storage occupied by a display processor jump command	198
S_p	Storage occupied by a pointer	197
S_r	Storage occupied by a display processor subroutine return command	202

Symbol	Meaning	Page
S_s	Storage occupied by a display processor subroutine call command	198
S_t	Storage required to designate block type when the computer performs Algorithm 3.1	196
S_t'	Storage required to designate block type when the display processor performs Algorithm 3.1	196
T_b	Time required to determine whether a block represents an element of B, $E/D\text{-}E_0/D_0$, or E_0/D_0	175
T_c	Time required to compare two pointers	166
T_{df}, T_{dg}	Total times required to plot entities in E_0	210
T_j	Time required to execute a display processor jump command	207
T_j'	Time required for the computer to interpret a display processor jump command as a pointer	171
T_p	Time required to extract a pointer from a known position in the structure	166

Symbol	Meaning	Page
T_s	Time required to execute a display processor subroutine call command	208
T_s'	Time required for the computer to interpret a display processor subroutine call command as a pointer	176
T_{tf}, T_{tg}	Total times (in addition to memory access time) required for the display processor to perform coordinate transformations while the picture is generated once	211

Other frequently used symbols:

g	Total time during which the computer is devoted to the picture generation process while the picture is generated once	141
ℓ_i	The ith basic operation	144
$\bar{r}$	Cost	142
s	Total storage occupied by the structure	94
t_f	Time which elapses while the picture is generated once	150
u_i	The ith design decision	180

Chapter 1

INTRODUCTION

The objective of this research is to describe the topological structure of an interactive computer display picture and to compare the time which is required to operate on various implementations of this structure in response to user inputs. The scope of the work may be clarified with the aid of Figure 1.1, which represents a simplified form of the general interactive computer display program as described by W. R. Sutherland [70]. In this figure, the rectangles represent sections of the program, the circles represent sets of data, and the directed paths represent flow of information. Each input which affects the picture is an element of a one-dimensional control language, which consists of sequences of light pen motions, push button hits, etc. In response to these inputs, the control language interpreter modifies the structure of the picture which is being displayed. The resultant structure is then further modified by a constraint program which compensates for the coarseness of the control language inputs, and it is interpreted by the picture generator to produce a new picture. (Typically, it is also interpreted to control an analysis program.) This paper is concerned with the topological structure which is interpreted to generate the picture, the picture generator, and the picture which is produced.

1.1 Brief History of Topological Structures and Display Equipment

The need for representing the structure of a picture, in addition to its appearance, was originally demonstrated by I. E. Sutherland's

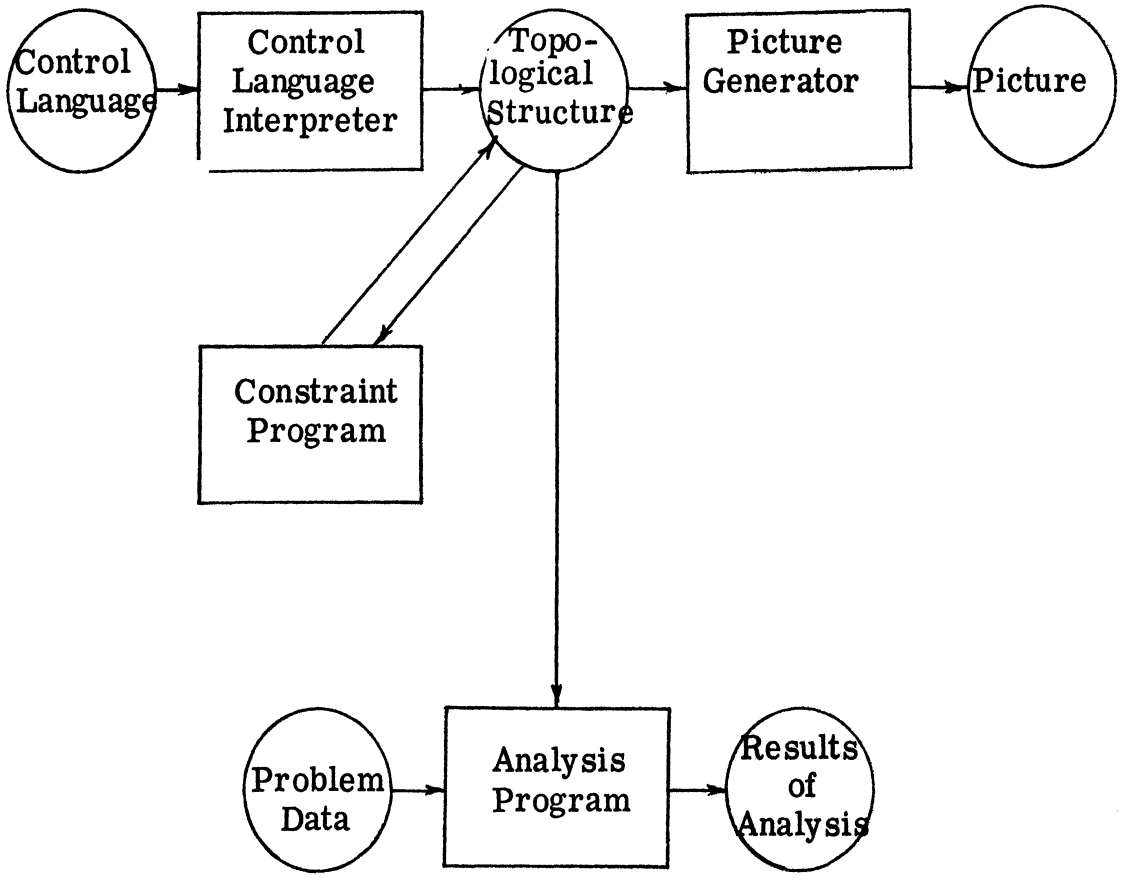


Figure 1.1

Interactive Computer Display Program
as Described by W. R. Sutherland

SKETCHPAD [67, 68] ,which is generally considered to be the first major effort to support an interactive computer display [63 p. 58] . This program was designed to produce two-dimensional line drawings in response to push button and light pen inputs. Since all operations were purely graphical in nature, SKETCHPAD did not include the analysis program shown in Figure 1.1. However, it did include a facility for displaying multiple copies of a subpicture, as well as a program which enforced geometric constraints. The topology of the picture*, as well as information which described constraints, was stored in the form of a list structure. This structure facilitated both the display of multiple copies of a subpicture and the identification of those other subpictures which would be modified whenever a subpicture was either referenced by a demonstrative control language input or modified by the enforcement of a constraint.

Soon after SKETCHPAD was implemented, interest developed in improving communication between man and machine through the use of interactive computer displays. The first efforts concentrated on the establishment of more sophisticated hardware, as exemplified by the SCHOOLER display [4] , Graphic-1 at Bell Telephone Laboratories [47] , and DAC-I at General Motors [15] . Not much thought was given to the structure of pictures to be displayed by this equipment, although considerable thought was given to the type of software capabilities which were desirable from the user's point of view [2,15, 29,47] .

*Throughout this paper, the word "topology" is used for historical reasons. This word is not to be interpreted in a mathematical context.

Because time-sharing systems were becoming available, the new hardware included a programmable display processor which either had its own buffer memory or shared the memory of a small general purpose computer in order to refresh its picture. However, this equipment required the support of a large computer (e.g., time-sharing facility) for analysis programs and, possibly, some or all of the programs which operated on the topological structure of the picture.

After some preliminary experimentation with the new type of equipment, a software system which provided for the representation of the topology of a picture on this equipment was developed at Bell Telephone Laboratories for use with GRAPHIC-2 [12]. Each primitive subpicture (i.e., each subpicture which was composed of not more than one part) was represented by a display processor program which was called a "leaf", and these leaves were referenced by pointers in a list structure which described other subpictures in terms of those represented by leaves. Coordinates were also stored in the list structure such that the coordinates at which each copy of each primitive subpicture should be displayed could be obtained by summing these coordinates along a path through the structure. The display processor was driven interpretively from the list structure by the associated general purpose computer, which computed the coordinates of each copy of each primitive subpicture from the structure, and then allowed the display processor to execute the corresponding leaf, starting at the computed coordinates.

The structures used by SKETCHPAD and GRAPHIC-2 were similar in that they both described picture topologies. However, pictures were generated from these two structures in quite different fashions. SKETCHPAD interpreted a list structure to produce a table of the coordinates of points to be plotted on the display screen. Then the program generated the picture to be displayed by transferring the coordinates of each point represented in the table to the display processor. Consequently, the SKETCHPAD picture generator was essentially a program. However, in GRAPHIC-2, the general-purpose computer program interpreted the list structure only to compute a pair of coordinates at which a primitive subpicture was to be displayed and to locate a display processor program (leaf) for that subpicture. The display processor itself then executed the leaf and produced the subpicture at the computed coordinates. Consequently, the GRAPHIC-2 picture generator was partly a program and partly display processor hardware.

A third form of a topological structure was implemented previously by the author [26]. This structure was implemented entirely as a display processor program. Consequently, the picture generator for this structure was entirely display processor hardware.

Recent effort in the design of display processors has yielded devices which interpret 3-dimensional data and plot perspective projections of this data on 2-dimensional screens. Examples of such

devices are the Adage Graphics Terminal [71] and the Evans and Sutherland LDS-1 [18]. The latter of these devices is sufficiently versatile to generate a picture from a topological structure without the aid of a general-purpose computer program.

1.2 Types of Implementation

As evidenced by the above examples, the picture generator may be either a program, a combination of a program and display processor hardware, or entirely display processor hardware. No one of these schemes has a clear advantage over the others for all applications. A software point-plotting picture generator, such as the one which was employed by SKETCHPAD, can be implemented with even the most primitive display processor hardware. However, this scheme requires storage both for the structure of the picture and for a display buffer. Furthermore, unless the picture is produced on a storage tube or some similar device, much computation time is consumed merely maintaining a static picture. Less computation time is required for this process if a scheme such as that used in GRAPHIC-2 is employed. Furthermore, no storage is required for a display buffer. However, this scheme requires that a data channel be available to send information to the display processor and that the display processor include a display generator which is capable of producing some basic geometric forms, such as vectors, which may be displayed relative to given coordinates. Unlike the software point-plotting technique, this technique often limits

the flexibility with which copies of a subpicture may be displayed. For example, a single leaf could not be executed so that copies of its corresponding subpicture would be displayed at various angles of rotation in GRAPHIC-2. Even more limitations are imposed if the display processor hardware itself is to be used as the picture generator. In particular, the words which appear in the topological structure must be executable display processor commands. Furthermore, if this procedure is used and demonstrative control language inputs are to be identified, the display processor must contain a subroutining facility which saves return addresses on a push-down stack which may be examined by the general-purpose computer. However, this procedure has the advantage over the others that no computation time is required to maintain a static picture.

1.3 Types of Structure

Except for the constraints represented in the SKETCHPAD list structure, each of the structures in the three examples given above describes only the topology of the picture and drawing information for primitive subpictures. This type of structure is common, for it has also been applied to other display systems, e.g., SKETCHPAD III [28], DIM [5], and PLAN-PGS [11]. However, certain other forms of information appear to be suitable for inclusion in the structure, as they simplify certain operations on the picture. In particular, A. C. Shaw[61, 62] has developed a formal description of picture parsing which provides some insight into the type of information which

might be included. Although this description was developed primarily for pattern recognition purposes, the facility which it provides for representing concatenation of subpictures is often a useful part of a structure from which a picture is to be generated.

1.4 Objectives

In the following chapter, the information which defines a topological structure is discussed. An original contribution of this chapter is a means of representing coordinate transformations such that subpictures are effectively concatenated in a manner similar to that described by Shaw. To demonstrate the feasibility of this method, a program which employs a topological structure in which coordinate transformations are represented in this way was written by the author and is described in Appendix A.

In Chapter 3, the average time which is required to operate on an implementation of the structure is parameterized as a cost function. The discussion in this chapter illustrates the major decisions which affect the choice of implementation. This cost function is then expressed as a function of these decisions in Chapter 4 and is applied to several applications in Chapter 5. The set of programs which were used to study these applications is described in Appendix B. Finally, suggestions for future extensions to this effort are discussed in Chapter 6.

Chapter 2

THE TOPOLOGICAL STRUCTURE

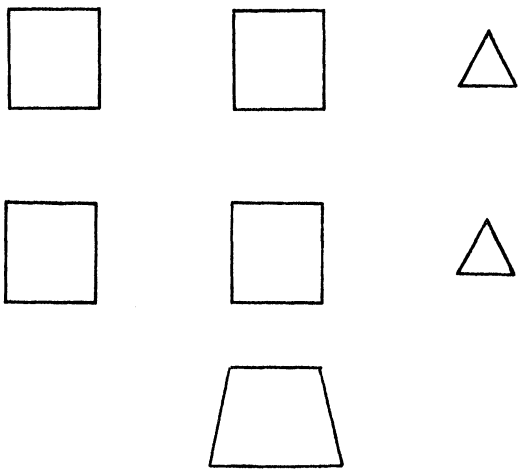
In order to compare various implementations of topological structures, the information which is contained in these structures must be defined. Generally, a topological structure will be considered to describe both a topology of picture parts and the coordinate transformations which are necessary to display these picture parts. In this chapter, a topological structure which has been used widely in previous interactive computer graphics programs (e.g., SKETCHPAD [67, 68] , SKETCHPAD III [28] , GRAPHIC-2 [12] , DIM[5] , PLAN-PGS [11] , PENCIL [74]) is described. This structure provides for the independent modification of picture parts which are not topologically related. A second structure which has been developed by the author is then described. This latter structure provides for the enforcement of concatenation constraints among picture parts as a consequence of refreshing the picture.

The usefulness of representing the topology of the picture was originally demonstrated by SKETCHPAD [67, 68] . As mentioned in the previous chapter, one use of this information was to permit the display of multiple copies of a subpicture. The ability to display multiple copies of a subpicture eliminated the necessity to store individual copies of identical subpictures. Consequently, since the storage required for topology information was small relative to that required to

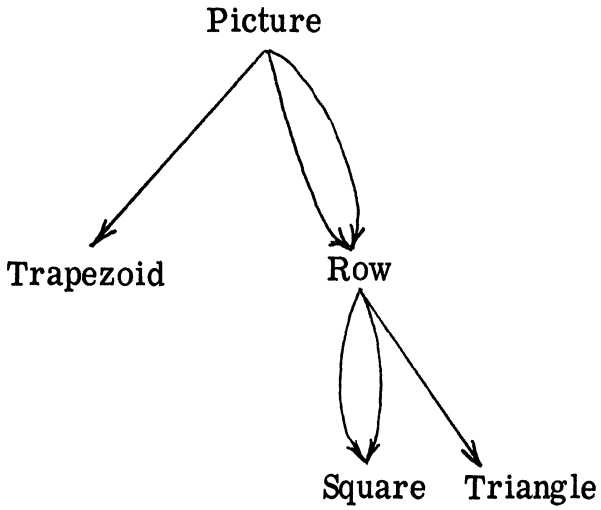
store copies of subpictures, the representation of the topology of the picture in memory resulted in a reduced total storage requirement. Furthermore, all picture parts which were represented as copies of a common subpicture could be modified simultaneously by modifying the subpicture.

Multiple copies of a subpicture may be displayed by interpreting a topological structure which uses copies of this subpicture as parts of a larger subpicture which is displayed. This procedure may be illustrated with the aid of Figure 2.1. Figure 2.1a represents a picture to be displayed and Figure 2.1b shows a representation of the topology of this picture. In Figure 2.1b, each directed line indicates that a copy of the subpicture to which it points is used as part of the subpicture at its tail. For this example, the picture consists of a trapezoid and two rows of objects, each of which consists of two squares and a triangle. The subpictures which are represented are the trapezoid, the square, the triangle, the row, and the entire picture. Only three of these subpictures are used to generate polygons in the picture: the trapezoid, the square, and the triangle. However, seven polygons appear in the picture because four copies of the square are displayed and two copies of the triangle are displayed.

In order to use a copy of a subpicture as part of a larger subpicture, some information about how the smaller subpicture is to appear in the larger subpicture is needed. In SKETCHPAD, this



a. Picture



b. Representation of Topology

Figure 2.1

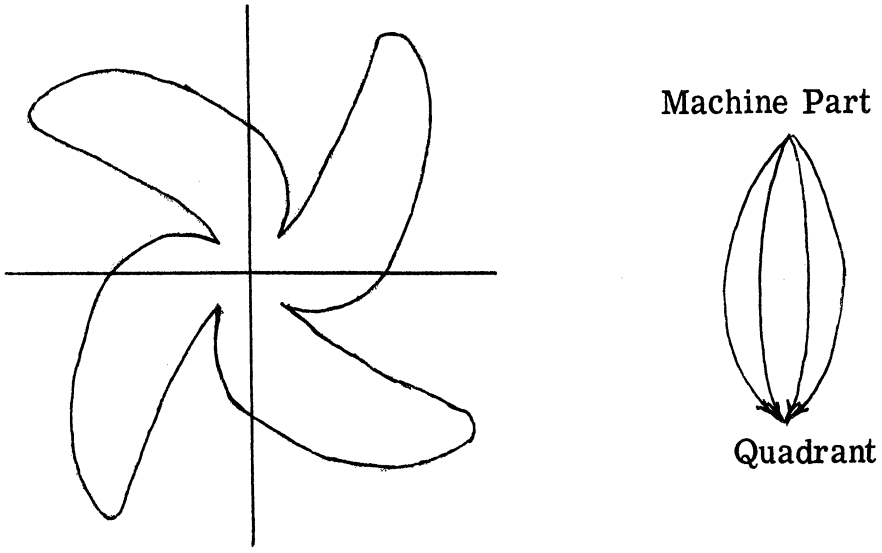
A Picture and a Representation of Its Topology

information was represented by four variables: x and y coordinates, a rotation angle, and a scale factor. (The values actually stored were the x and y coordinates and the products of the scale factor by the sign and cosine of the angle of rotation.) These four variables represented a transformation of coordinates to be applied to each point in the smaller picture when it was displayed. A set of values for these four variables, together with a subpicture, was called an instance. A subpicture could then be defined in terms of smaller subpictures as a collection of instances.

For the example in Figure 2.1, there are two instances of the square, one instance of the trapezoid, one instance of the triangle, and two instances of the row. Each of these instances is represented as a directed line in Figure 2.1b. The number of instances of a subpicture is not necessarily the number of times that that subpicture appears in the overall picture. In this example, there are four squares in the picture, yet there are only two instances of the square. Likewise, the variables which help define an instance are not measured relative to the overall picture. In this example, the coordinates of the two instances of the square and the instance of the triangle are measured relative to the coordinates of each row.

As mentioned above, the topology of the picture is also used to identify those other subpictures which are modified whenever a subpicture is referenced by a demonstrative control language input (i.e.,

a. A Resistor from an Electrical Network Program



b. A Machine Part from a Drafting Program

Figure 2.2

Examples of Pictures for Which the Topology of Each Picture Facilitates Response to Control Language Inputs

an input which consists of identifying a picture part by pointing to it with a light pen, stylus, etc.), Some of the applications of the topology of the picture to this problem may be illustrated with the aid of Figure 2.2. Figure 2.2a shows a resistor symbol as it might appear in a circuit analysis program. The symbol is considered to be composed of two ports, i.e., attachment points for connections to other elements, a parameter value, and a body. In the picture, each port appears as a small circle, the parameter value appears as the number 1.0, and the body appears as the rest of the symbol. For purposes of illustration, assume that a program is running which allows the user to point at the symbol with a light pen and either (1) change the value of the parameter to a value which has just been inputted on some other device, or (2) move the symbol with the light pen if no parameter value has been inputted. In either case, the light pen may reference any one of several primitive subpictures: the port, the parameter, or the body of the symbol. The resistor subpicture is identified as the subpicture which consists of a set of instances of primitive subpictures and which was being displayed at the time of the light pen hit. In the former case, the parameter subpicture to be replaced is identified as the primitive subpicture which is part of the resistor subpicture and which represents a parameter. (Whether or not a subpicture represents a parameter is determined from information external to the topology, e.g., from an ordering imposed on the parts of the resistor subpicture.) In the latter case,

the coordinates associated with the instance of the resistor subpicture which was referenced with the light pen are modified in order to move the symbol.

Figure 2.2b represents a picture which maintains symmetry about the origin of the two axes shown. The program is assumed to be capable of modifying only the quadrant subpicture in response to control language inputs. The machine part subpicture is displayed as four instances of this quadrant subpicture at angles 0, $\pi/2$, π, and $3\pi/2$. Not only does the topological structure provide the information necessary to identify what other changes should be made to the picture when a quadrant is modified, but it guarantees that the other quadrants will be correspondingly modified without intervention by the program.

2.1 Topology

In order to discuss the topology of a picture and the representation of this topology for interactive computer display applications, the following definition is needed:

Definition 2.1

An entity is either a part of the displayed picture* which is interpreted as a unit by the program or the entire displayed picture.

---

*The displayed picture is the picture which is transferred to the display processor, regardless of whether or not all of it is visible to the user. For example, the clipping divider [64] developed by Evans and Sutherland Corporation may be used to select only part of the displayed picture to be plotted on a display screen.

The entities in the picture shown in Figure 2.1 are the four squares, the two triangles, the trapezoid, the two rows, and the entire picture. Generally, entities are chosen so that they have a semantic meaning to the user. For example, the user who observes the picture in Figure 2.1a immediately recognizes the seven polygons. The fact that six of these polygons are arranged to form two rows may not be so obvious to him. However, if he is interacting with the program, he will generally recognize that the rows are entities by observing responses to his inputs. For example, he may generate inputs which cause each row to be moved as a unit.

With the term "entity" defined, the terms "subpicture" and "instance" may be clarified by the following definitions:

Definition 2.2

A subpicture is a picture to which a coordinate transformation may be applied to produce an entity.

Definition 2.3

An instance is a part of a subpicture which is obtained by applying a coordinate transformation to a smaller subpicture.

Five subpictures are involved in Figure 2.1: a square, a triangle, a trapezoid, a row, and the entire picture. The row subpicture consists of two instances of the square subpicture and one instance of the

triangle subpicture, and the entire picture subpicture consists of two instances of the row subpicture and one instance of the trapezoid subpicture. The entire picture entity will be considered to be the same as the entire picture subpicture, i.e., the coordinate transformation which is applied to the entire picture subpicture in order to produce the entire picture entity is the identity transformation.

2.1.1 Definition of Topology

The topology of the picture may be described by a relation C which identifies which entities are parts of other entities. if E is the (finite) set of all entities which are represented in the picture, C may be defined on E as follows:

$$C = \{(e, e') \mid (e, e') \in E \times E \text{ and } e' \text{ is a part of } e\}. \qquad (2.1)$$

The word "part" is interpreted to mean "something less than the whole" (which is not null), and the partitioning of each entity into its component parts is a function of the application of the graphics program. Because a part of an entity is strictly less than the entire entity, C is irreflexive, i.e.,

$$e \in E \Rightarrow (e, e) \notin C. \qquad (2.2)$$

Furthermore, since a part of a part of an entity is itself a part of that entity, C is also transitive, i.e.,

$$(e, e') \in C \text{ and } (e', e'') \in C \Rightarrow (e, e'') \in C. \qquad (2.3)$$

A third property of C may be specified by considering the entity $\lambda \in E$, which represents the entire picture. Since all entities are parts of this picture, λ is related to every other element $e \in E$ by C, or

$$e \in E \text{ and } e \neq \lambda \Longrightarrow (\lambda, e) \in C. \qquad (2.4)$$

A further property of C may be derived from the fact that C is both irreflexive and transitive. However, before this property can be stated, the following definition and theorem must be stated:

Definition 2.4

A directed cycle of length n ($n \geq 1$) which is formed by a relation Y on the set X is a sequence $(x_1, x_2, \ldots, x_n) \in X^n$ such that $(x_i, x_{i+1}) \in Y$ for $i=1, 2, \ldots, n-1$ and $(x_n, x_1) \in Y$.

Theorem 2.1

A transitive and irreflexive relation forms no directed cycles.

Proof

Assume that Y is a transitive and irreflexive relation on a set X which forms the directed cycle $(x_1, x_2, \ldots, x_n) \in X^n$. Then $(x_i, x_{i+1}) \in Y$ for $i=1, 2, \ldots, n-1$, and $(x_n, x_1) \in Y$. (x_n, x_k) is shown to be an element of Y for $k=1, 2, \ldots, n$ by induction as follows:

Basis: $k=1$. $(x_n, x_1) \in Y$ from the definition of a directed cycle.

Induction step: Assume that $(x_n, x_i) \in Y$ for some $i \leq n-1$.

Since $i \leq n-1$, $(x_i, x_{i+1}) \in Y$. Since Y is transitive, $(x_n, x_i) \in Y$ and $(x_i, x_{i+1}) \in Y \Longrightarrow (x_n, x_{i+1}) \in Y$.

In particular, $(x_n, x_n) \in Y$. However, $(x_n, x_n) \notin Y$ because Y is irreflexive. ∎

Since C is both irreflexive and transitive, C forms no directed cycles.

Although the relation C properly describes the topology of the picture, it is generally a rather large set, and, consequently, storage of its elements in a general-purpose computer may not be practical. However, by taking advantage of the fact that C is transitive and forms no directed cycles, a relation $A \subset C$, which generally has fewer elements than C, may be defined such that C may be easily generated from A. The relation A is defined as follows:

$$A = \{(e,e') \mid (e,e') \in C \text{ and } \nexists\ e'' \in E \text{ such that } (e,e'') \in C \text{ and } (e'',e') \in C\}. \tag{2.5}$$

C is then the transitive closure of A, i.e., C may be generated from A by adding to A those elements which are necessary to make it transitive.

The definition of A given above is useful only because C forms no directed cycles. If C did form a directed cycle, the elements associated with the directed cycle could not be generated from A. For example, assume that C was transitive and consisted only of those elements necessary to form the directed cycle (e_1, e_2, e_3), i.e.,

$$C = \{(e_1, e_1), (e_2, e_2), (e_3, e_3), (e_1, e_3), (e_3, e_2), (e_2, e_1), (e_1, e_2),$$

$$(e_2, e_3), (e_3, e_1)\}. \qquad (2.6)$$

For this relation, $A = \phi$, and the transitive closure of A is ϕ.

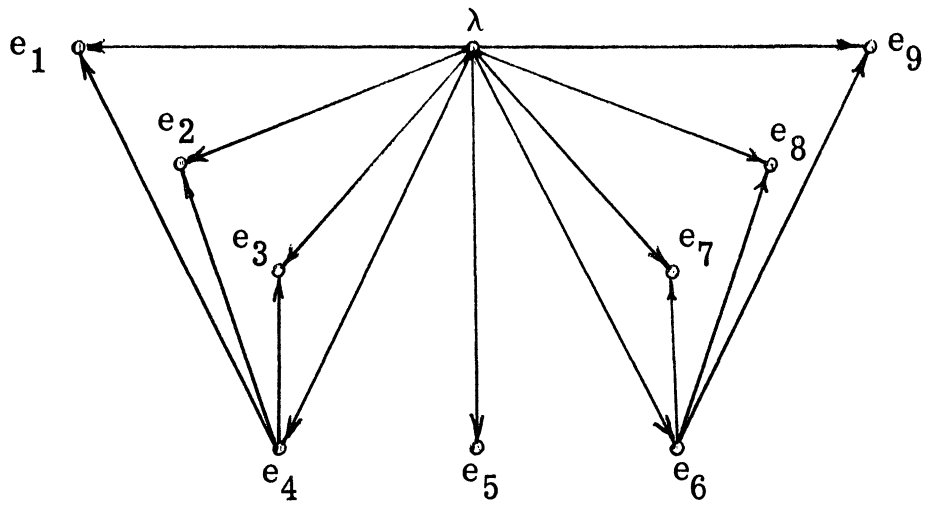
Graphs of the relations C and A for the example picture shown in Figure 2.1 are shown in Figure 2.3. The graph of A commonly assumes the form of a tree, as it does in this example. This occurs whenever each entity in the picture is not considered to be a part of two or more entities which are not themselves related by C.

However, the facility to represent a relation A whose graph does not assume the form of a tree is sometimes quite useful. In particular, the connectivity of a network diagram (i.e., a diagram in which element symbols are to be associated with each other via connections, such as electrical circuit diagrams flow charts, bridge truss diagrams, etc.) may be represented through the use of this facility. For example, Figure 2.4 shows an electrical circuit diagram and the graph of a relation A for this diagram. Each port of each element in the network is represented by a connection which is considered to be a part of that element. (An unconnected port is considered to be connected to itself.) Two ports are connected to each other if and only if they are represented by the same connection.

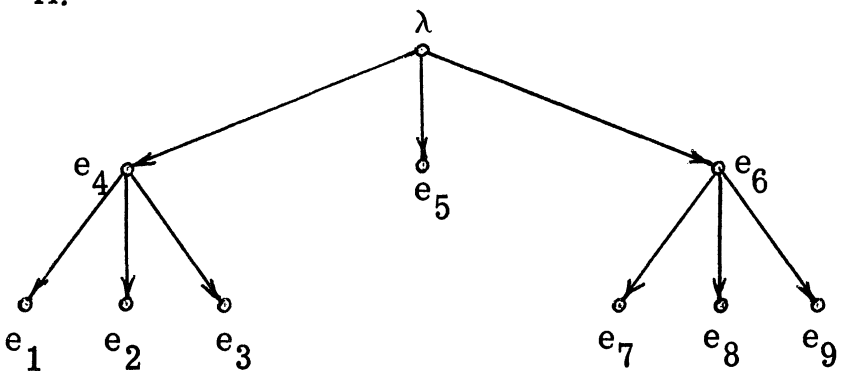
2.1.2 Dependent Representation of Entities

Even if A, rather than C, is stored, some information may be stored redundantly. In particular, entities which are described as

C:



A:



Trapezoid: e_5

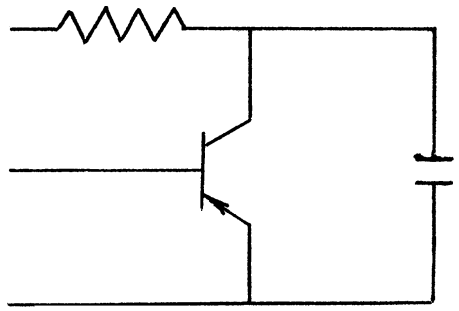
Squares: e_1, e_2, e_7, e_8

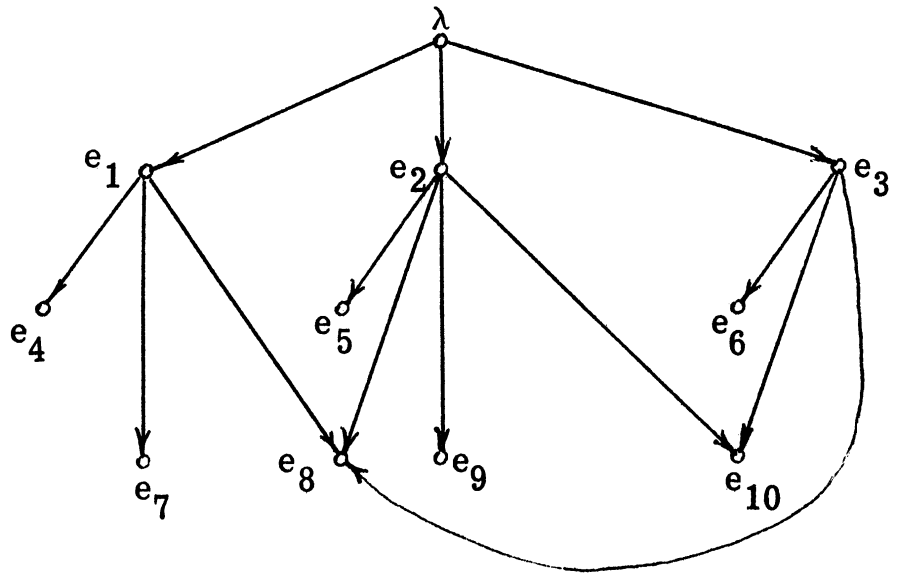
Triangles: e_3, e_9

Rows: e_4, e_6

Figure 2.3

Graphs of the Relations C and A for Picture in Figure 2.1





Elements: e_1, e_2, e_3

Resistor symbol: e_4

Transistor symbol: e_5

Capacitor symbol: e_6

Connections: e_7, e_8, e_9, e_{10}

Figure 2.4

An Electrical Circuit Diagram and the Representation of Its Connectivity by the Relation A

collections of other entities by similar structures and which differ only by a transformation of coordinates need not be stored independently.

2.1.2.1 Structural Equivalence

An equivalence relation $T \subset E \times E$ will be defined to classify those entities which have similar structures. In order to define the relation T, the function $S : E \rightarrow \mathcal{P}(E)$ is first defined, where $\mathcal{P}(E)$ is the power set of E, i.e.,

$$\mathcal{P}(E) = \{X \mid X \subset E\}. \tag{2.7}$$

S is defined as follows:

$$S(e) = \{e' \mid (e, e') \in A\}. \tag{2.8}$$

This function maps each entity into the set of its immediate successors under the relation A. S may now be applied to define the subsets $E_0, E_1, E_2, \ldots$ of E as follows:

$$E_0 = \{e \mid S(e) = \phi\}$$

$$E_i = \{e \mid e \notin \bigcup_{j=0}^{i-1} E_j \text{ and } S(e) \subset \bigcup_{j=0}^{i-1} E_j\}, \quad i=1, 2, \ldots$$

A finite number of these subsets are to represent a partitioning of the set E. For this to be true, there must exist an integer m such that E is the union of $E_0, E_1, E_2, \ldots, E_m$ and $E_i \cap E_j = \phi$ for $i \neq j$ and $0 \leq i, j \leq m$. From the definition of $E_0, E_1, E_2, \ldots$ the latter condition is clearly satisfied for all nonnegative integers m. The former condition is verified by the following theorem:

Theorem 2.2

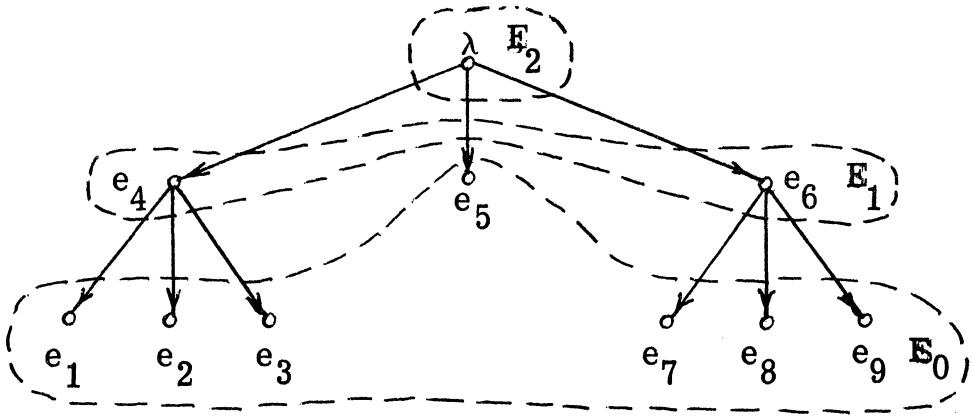
There exists an integer $m \geq 0$ such that $E = \bigcup_{i=0}^{m} E_i$.

Proof

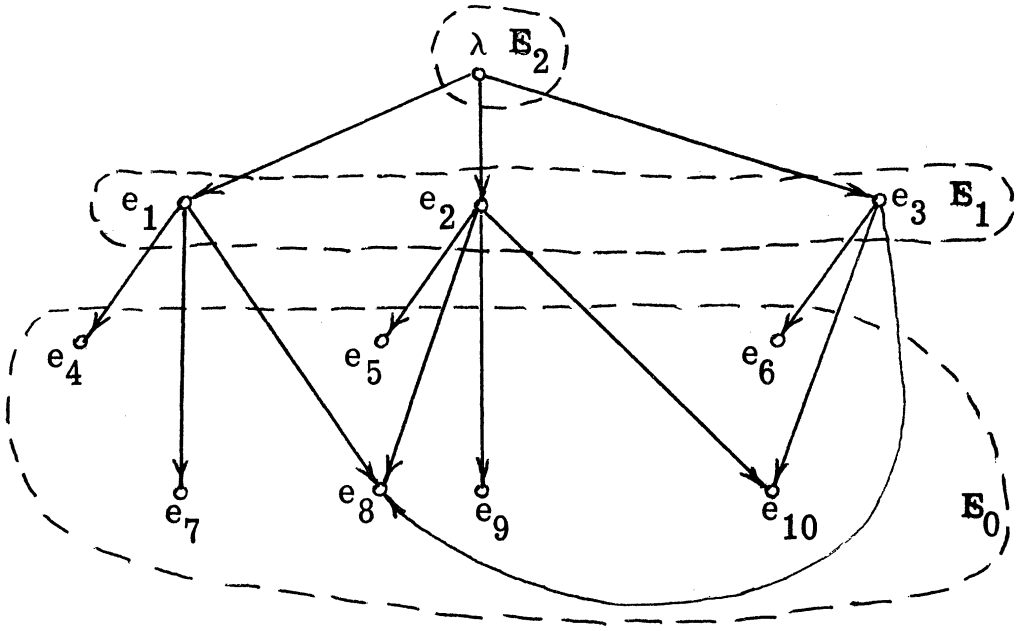
Since E is a finite set and the subsets $E_0, E_1, E_2, \ldots$ are defined to be disjoint, the integer m exists if it can be shown that $E \neq \bigcup_{j=0}^{i-1} E_j \Longrightarrow E_i \neq \phi$ for $i=1, 2, \ldots$. Assume that $E \neq \bigcup_{j=0}^{i-1} E_j$ for some integer $i > 1$. Then, $\lambda \in E - \bigcup_{j=0}^{i-1} E_j$. Since $\lambda \notin \bigcup_{j=0}^{i-1} E_j$ and $(\lambda, e) \in C$ for all $e \in E$, there exists at least one element $e \in E_{i-1}$ such that $(\lambda, e) \in C$. Consequently, there exists an element $e' \in E_i$ such that $(e', e) \in A$, i.e., $E_i \neq \phi$. ∎

As a result of this theorem, the only E_i's which will be considered further are $E_0, E_1, \ldots, E_m$.

For both the example in Figures 2.1 and 2.3 and the example in Figure 2.4, $m = 2$. The subsets E_0, E_1, and E_2 for these two examples are depicted in Figure 2.5. The equivalence classes of the relation T are formed by partitioning the subsets $E_0, E_1, E_2, \ldots, E_m$. For the example in Figure 2.5a, the equivalence classes of T are simply the subsets E_0, E_1, and E_2. However, for the example in Figure 2.5b, the equivalence classes of T are E_0, E_2, and two sets which are formed by partitioning E_1.



a. Example from Figures 2.1 and 2.3



b. Example from Figure 2.4

Figure 2.5

Examples of the Subsets $E_0, E_1, E_2, \ldots, E_m$

As a further step toward defining the relation T, the relations $T \subset E_i \times E_i$ are defined for $i = 0,1,2,\ldots,m$ as follows:

$$T_0 = E_0 \times E_0 \tag{2.11}$$

$T_i = \{(e,e') \mid (e,e') \in E_i \times E_i$ and there exists a one-one and onto function $\alpha: S(e) \longrightarrow S(e')$ such that

$$e'' \in S(e) \Longrightarrow (e'', \alpha(e'')) \in \bigcup_{j=0}^{i-1} T_j\}, \quad i=1,2,\ldots,m. \tag{2.12}$$

T is then defined to be the union of these relations, i.e.,

$$T = \bigcup_{i=0}^{m} T_i. \tag{2.13}$$

As mentioned above, T is required to be an equivalence relation. In order to show that T is an equivalence relation, the relations $T_0, T_1, T_2, \ldots, T_m$ are first shown to be equivalence relations by the following theorem:

Theorem 2.3

$T_0, T_1, T_2, \ldots, T_m$ are equivalence relations.

Proof

$T_0, T_1, T_2, \ldots, T_m$ must be shown to be reflexive, symmetric, and transitive.

1. $T_0, T_1, T_2, \ldots, T_m$ are shown to be reflexive by induction as follows:

Basis: $e \in E_0 \Longrightarrow (e,e) \in E_0 \times E_0 \Longrightarrow (e,e) \in T_0 \Longrightarrow T_0$ is reflexive.

Induction step: Assume that $T_0, T_1, T_2, \ldots, T_i$ are reflexive. Let $e \in E_{i+1}$ and $\alpha: S(e) \longrightarrow S(e)$ be the identity function. Then, $S(e) \subset \bigcup_{j=0}^{i} E_j$, and, since T_j is reflexive for $j \leq i$, $e' \in S(e) \Longrightarrow e' \in \bigcup_{j=0}^{i} E_j \Longrightarrow (e',e') \in \bigcup_{j=0}^{i} T_j \Longrightarrow (e', \alpha(e')) \in \bigcup_{j=0}^{i} T_j$. Consequently, $(e,e) \in T_{i+1}$ and T_{i+1} is reflexive.

2. $T_0, T_1, T_2, \ldots, T_m$ are shown to be symmetric by induction as follows:

Basis: $(e,e') \in T_0 \Longrightarrow (e,e') \in E_0 \times E_0 \Longrightarrow (e',e) \in E_0 \times E_0 \Longrightarrow (e',e) \in T_0 \Longrightarrow T_0$ is symmetric.

Induction step: Assume that $T_0, T_1, T_2, \ldots, T_i$ are symmetric. Let $(e,e') \in T_{i+1}$. Then, there exists a one-one and onto function $\alpha: S(e) \longrightarrow S(e')$ such that $e'' \in S(e) \Longrightarrow (e'', \alpha(e'')) \in \bigcup_{j=0}^{i} T_j \Longrightarrow (\alpha^{-1}(\alpha(e'')), \alpha(e'')) \in \bigcup_{j=0}^{i} T_j$. Since T_j is symmetric for $j \leq i$, $e'' \in S(e) \Longrightarrow \alpha(e'' \in S(e') \Longrightarrow (\alpha(e''), \alpha^{-1}(\alpha(e''))) \in \bigcup_{j=0}^{i} T_j$. Consequently, there exists the one-one and onto function $\alpha^{-1}: S(e') \longrightarrow S(e)$ such that $e''' \in S(e') \Longrightarrow (e''', \alpha^{-1}(e''')) \in \bigcup_{j=0}^{i} T_j$. Then $(e',e) \in T_{i+1}$ and T_{i+1} is symmetric.

3. $T_0, T_1, T_2, \ldots, T_m$ are shown to be transitive by induction as follows:

Basis: $(e, e') \in T_0$ and $(e', e'') \in T_0 \Longrightarrow (e, e'') \in E_0 \times E_0$ $\Longrightarrow (e', e'') \in T_0 \Longrightarrow T_0$ is transitive.

Induction step: Assume that $T_0, T_1, T_2, \ldots, T_i$ are transitive. Let $(e, e') \in T_{i+1}$ and $(e', e'') \in T_{i+1}$. Then, there exists a one-one and onto function $\alpha: S(e) \longrightarrow S(e')$ such that $e''' \in S(e) \Longrightarrow$ $(e''', \alpha(e''')) \in \bigcup_{j=0}^{i} T_j$, and there exists a one-one and onto function $\alpha': S(e') \longrightarrow S(e'')$ such that $e''' \in S(e) \Longrightarrow \alpha(e''') \in S(e') \Longrightarrow$ $(\alpha(e'''), \alpha'(\alpha(e'''))) \in \bigcup_{j=0}^{i} T_j$. Since T_j is transitive for $j \leq i$, there exists the one-one and onto function $\alpha' \alpha: S(e) \longrightarrow S(e'')$ such that $e''' \in S(e) \Longrightarrow (e''', \alpha' \alpha(e''')) \in \bigcup_{j=0}^{i} T_j$. Consequently, $(e, e'') \in T_{i+1}$ and T_{i+1} is transitive. ∎

A union of equivalence relations is not necessarily itself an equivalence relation. For example, consider the two equivalence relations

$$Y_1 = \{(x_1, x_2), (x_2, x_1), (x_1, x_1), (x_2, x_2)\}$$

and

$$Y_2 = \{(x_1, x_3), (x_3, x_1), (x_1, x_1), (x_3, x_3)\}.$$

$Y_1 \cup Y_2$ is not an equivalence relation, for it is not transitive, i.e., $(x_2, x_1) \in Y_1 \cup Y_2$ and $(x_1, x_3) \in Y_1 \cup Y_2$, but $(x_2, x_3) \notin Y_1 \cup Y_2$.

Consequently, Theorem 2.3 is not sufficient to establish that T is an equivalence relation. However, T may be shown to be an equivalence relation by considering both this theorem and Theorem 2.4 below.

Theorem 2.4

If $Y_i \subset X_i \times X_i$, $i=0, 1, 2, \ldots, n$, are equivalence relations and $X_i \cap X_j = \phi$ for $i \neq j$, then $\bigcup_{i=0}^{n} Y_i$ is an equivalence relation.

Proof

$\bigcup_{i=0}^{n} Y_i$ must be shown to be reflexive, symmetric, and transitive.

1. Since Y_i is reflexive for $i=0, 1, 2, \ldots, n$, $x \in \bigcup_{i=0}^{n} X_i \Longrightarrow$ there exists an integer k such that $x \in X_k \Longrightarrow (x, x) \in Y_k \Longrightarrow (x, x) \in \bigcup_{i=0}^{n} Y_i$. Consequently, $\bigcup_{i=0}^{n} Y_i$ is reflexive.

2. Since Y_i is symmetric for $i=0, 1, 2, \ldots, n$, $(x, x') \in \bigcup_{i=0}^{n} Y_i \Longrightarrow$ there exists an integer k such that $(x, x') \in Y_k \Longrightarrow (x', x) \in Y_k \Longrightarrow (x', x) \in \bigcup_{i=0}^{n} Y_i$. Consequently, $\bigcup_{i=0}^{n} Y_i$ is symmetric.

3. Assume that $(x, x') \in \bigcup_{i=0}^{n} Y_i$ and $(x', x'') \in \bigcup_{i=0}^{n} Y_i$. Then, there exists an integer j such that $(x, x') \in Y_j$, and there exists an integer k such that $(x', x'') \in Y_k$. $(x, x') \in Y_j$ and $(x', x'') \in Y_k \Longrightarrow x' \in X_j \cap X_k$. Since $X_j \cap X_k = \phi$ for $j \neq k$, $j = k$. Consequently, $(x, x'') \in Y_k \subset \bigcup_{i=0}^{n} Y_i$, and $\bigcup_{i=0}^{n} Y_i$ is transitive. ∎

From Theorem 2.3, $T_0, T_1, T_2, \ldots, T_m$ are equivalence relations. Furthermore, $T_i \subset E_i \times E_i$ for i=0, 1, 2, . . . , m, and $E_i \cap E_j = \phi$ for $i \neq j$ from the definition of E_i. Consequently, from Theorem 2.4, T is an equivalence relation.

Since T_0 is defined to be $E_0 \times E_0$, E_0 is always an equivalence class of T. Furthermore, since $e \in E$ and $e \neq \lambda \Longrightarrow (\lambda, e) \in C$, $E_m = \{\lambda\}$. Since E_m contains only one element, E_m is always an equivalence class of T.

For the example in Figure 2.5a, E_1 is readily seen to be an equivalence class of T. However, for the example in Figure 2.5b, E_1 is the union of two equivalence classes of T: $\{e_2\}$ and $\{e_1, e_3\}$. For these two examples, the two subgraphs of the relation A, one of which includes only the entity e and those entities e'' such that $(e, e'') \in C$ and the other of which includes only the entity e' and those entities e''' such that $(e', e''') \in C$, are isomorphic wherever $(e, e') \in T$. This condition is not true in general. In fact, there may not even exist a homomorphism between two such subgraphs, as illustrated by the example shown in Figure 2.6. In this example, $(e_1, e_2) \in T$, but there exists no homomorphism from one of the corresponding subgraphs to the other.

2.1.2.2 Entities as Transformed Subpictures

Although two entities which are related by T are structurally equivalent, they may appear as entirely different images in the picture. For example, the entities e_1 and e_3 in Figure 2.4 have

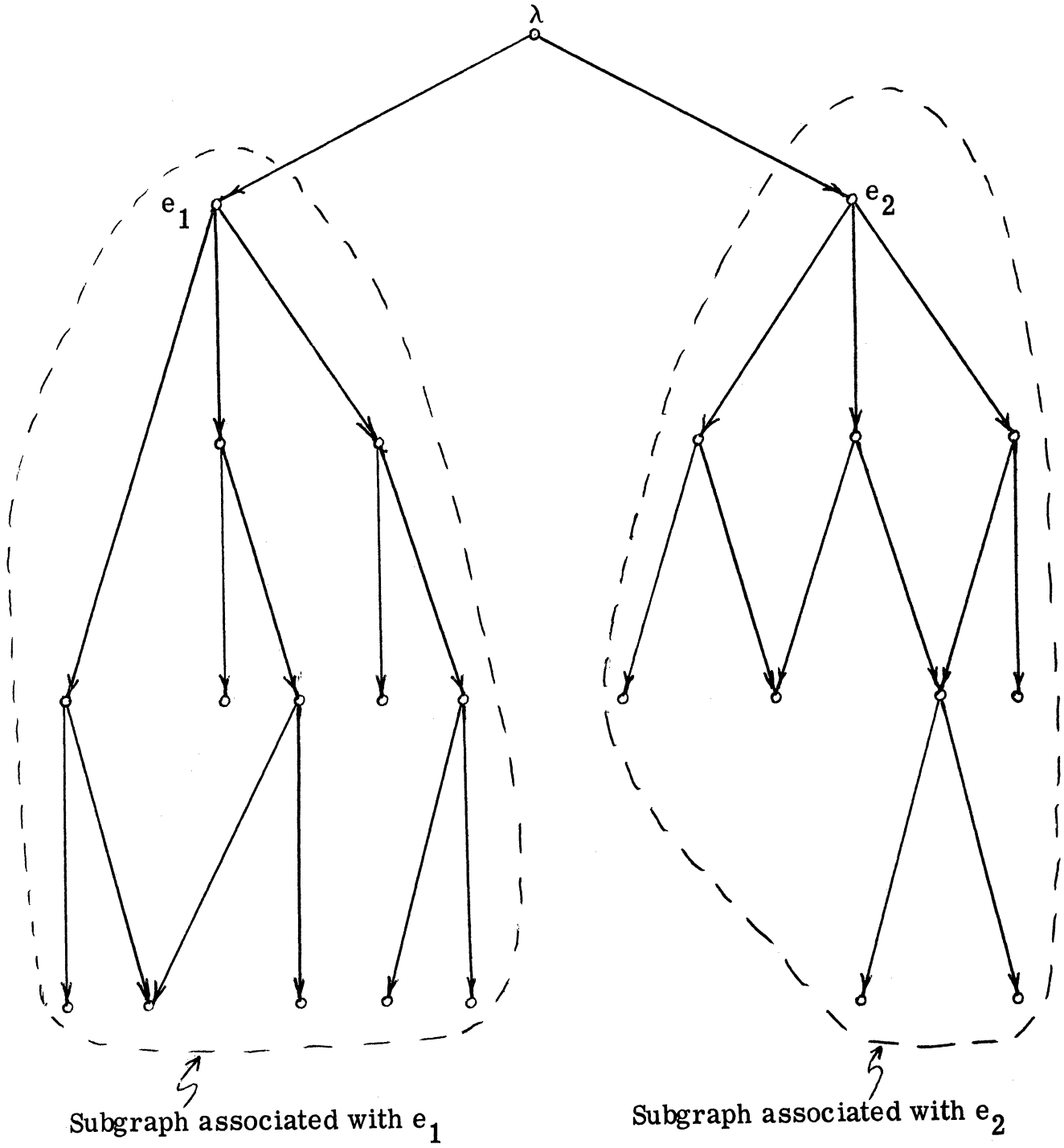


Figure 2.6

An Example of Two Equivalent Structures

equivalent structures, but one appears as a resistor in the picture, whereas the other appears as a capacitor. Such entities must be represented independently in storage. However, as mentioned above, if the two entities differ only by a transformation of coordinates, they need not be stored independently. In particular, the two entities may be displayed as copies of the same subpicture to which different coordinate transformations are applied. Two such entities are the two rows of polygons in Figure 2.1, which differ only by a translation of coordinates.

An equivalence relation $D \subset T$ may be defined to relate those entities which need not be stored independently. Before the relation D can be defined, however, a means of representing coordinate transformations must be described.

2.1.2.2.1 Representation of Coordinate Transformations

Coordinate transformations may be represented conveniently if the coordinates of each point in the picture are represented as homogeneous coordinates [56]. This representation of coordinates has been used previously in computer graphics programs, particularly those which provided for the manipulation of three-dimensional drawings [28, 57].

A point $(x_1, x_2, \ldots, x_n)$ in n-dimensional space may be represented by the homogeneous coordinates $(y_1, y_2, \ldots, y_{n+1})$ in (n+1)-dimensional space, where $y_i = y_{n+1}x_i$ for $i=1, 2, \ldots, n$.

Thus, $(x_1,x_2,\ldots,x_n,1)$ is one possible representation of the point $(x_1,x_2,\ldots,x_n)$. The coordinates $y_1,y_2,\ldots,y_n$ may be interpreted as direction numbers for a line in n-dimensional space which intersects the origin, where y_{n+1} describes the distance of a point on this line from the origin. Note that a point at infinity is described by choosing $y_{n+1} = 0$, whereas a point at the origin is described by choosing $y_i = 0$, $i = 1,2,\ldots,n$, and $y_{n+1} \neq 0$. A transformation may then be applied to these homogeneous coordinates to yield the homogeneous coordinates $(y_1',y_2',\ldots,y_{n+1}')$. Then the transformed coordinates $(x_1',x_2',\ldots,x_n')$ in n-dimensional space may be obtained according to the rule $x_i' = y_i'/y_{n+1}'$ for $i = 1,2,\ldots,n$.

Generally, a user of a computer display cannot comprehend pictures which are drawn in more than three dimensions. Consequently, no more than three dimensions will be considered further. Although most interactive computer display pictures are currently produced on two-dimensional screens, a three-dimensional picture may be represented in any one of several forms, such as: a set of three orthographic views [28], a halftone perspective projection [60,77], a pair of stereoscopic perspective projections [65], or a hologram [23,37].

A transformation of the coordinates of a point expressed with respect to the coordinate system W into the coordinates of this point expressed with repect to the coordinate system W' may be represented by the matric equation

$$v' = v\,H, \tag{2.14}$$

where v' is a vector which describes the homogeneous coordinates of the point with respect to W', v is a vector which describes the homogeneous coordinates of the point with respect to W, and H is a 4×4 matrix which describes the coordinate transformation. For example, a transformation of the coordinates of a point expressed with respect to W into coordinates expressed with respect to a coordinate system W' whose origin has the coordinates $(-\Delta x, -\Delta y, -\Delta z)$ in W, whose axes are parallel to the respective axes of W, and whose units of measurement are b times as large as those of W may be expressed as follows:

$$[\, wx' \; wy' \; wz' \; w \,] = [\, x \; y \; z \; 1 \,] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & b \end{bmatrix}$$

From this equation,

$$x' = \frac{x + \Delta x}{b}$$

$$y' = \frac{y + \Delta y}{b}$$

and

$$z' = \frac{z + \Delta z}{b} \; .$$

The transformation used in this example is actually a sequence of two simpler transformations. The coordinates of the point are first incremented, and then they are scaled. The transformation matrix H for

this example, then, is the product of two other transformation matrices which represent the simpler transformations. In this case,

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & b \end{bmatrix}.$$

A table of some simple transformation matrices is given in Figure 2.7.

2.1.2.2.2 Generation of Entities from Subpictures

In order to describe the generation of entities from subpictures, the function $\beta : E \longrightarrow Q$ is defined, where Q is a set of subpictures. $\beta(e)$ is the subpicture to which a coordinate transformation is to be applied to produce the entity e. Clearly, $\beta(e)$ is not unique, for the subpicture which is obtained by applying any non-singular coordinate transformation to $\beta(e)$ could also be used to generate e. However, in the interest of conserving storage, the values of β are usually chosen such that as many of them as possible are the same.

Suitable choices for the values of β for the picture represented in Figures 2.1 and 2.3 are shown in Figure 2.8. The coordinate system with respect to which each subpicture is drawn is also indicated in the figure. Any transformation which is applied to a subpicture to produce an entity is applied to coordinates expressed with respect to this coordinate system.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{bmatrix}$$

Translate. The origin of the new coordinate system is the point $(-\Delta x, -\Delta y, -\Delta z)$ in the original coordinate system. The respective axes of the two coordinate systems are parallel.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate. The origin of the new coordinate system is the origin of the original coordinate system. The x axes of the two coordinate systems are parallel. The angle between the y or z axes of the two coordinate systems is α.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & a \end{bmatrix}$$

Magnify all coordinates by the factor 1/a.

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

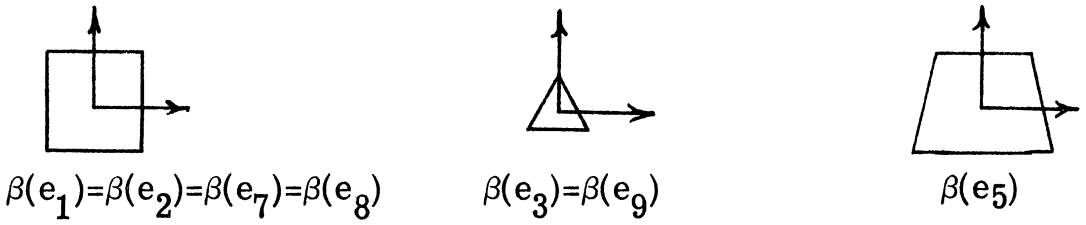
Magnify the x coordinate by the the factor a.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

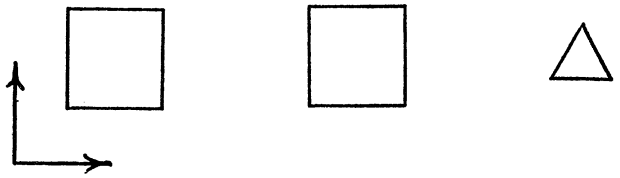
Form perspective projection on plane z = 0, such that the point of observation is z = -d.

Figure 2.7

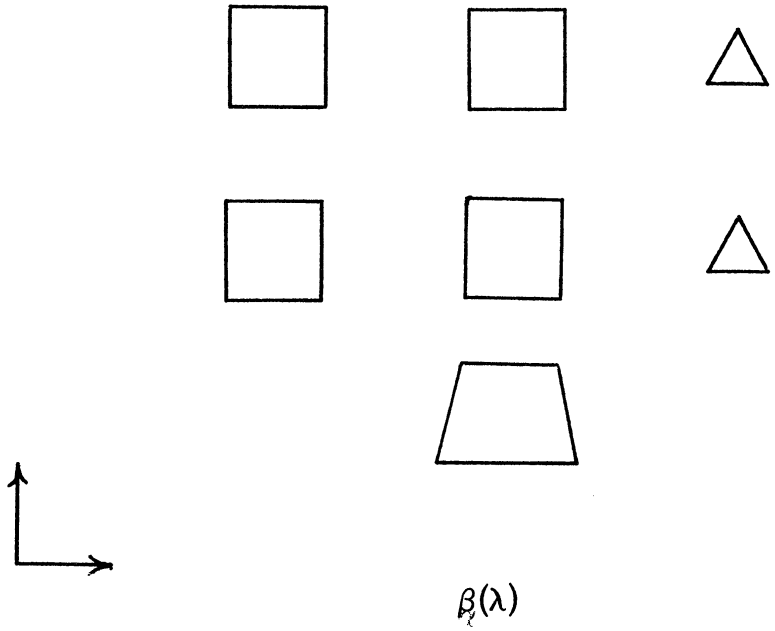
Some Simple Coordinate Transformation Matrices



$\beta(e_1)=\beta(e_2)=\beta(e_7)=\beta(e_8)$ $\beta(e_3)=\beta(e_9)$ $\beta(e_5)$



$\beta(e_4)=\beta(e_6)$



$\beta(\lambda)$

Figure 2.8

Subpictures Used to Generate Picture in Figure 2.1

The coordinate transformations which are applied to subpictures to generate entities will be restricted to be non-singular, and will be described by the function H: E —> M, where M is the set of non-singular 4×4 matrices. The only common transformation which is excluded by this restriction is perspective projection. However, the non-singular matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rather than the last matrix shown in Figure 2.7, may be used to represent prespective projection if only x and y coordinates are plotted. H(e) is then a matrix which represents the coordinate transformation which maps $\beta(e)$ into e. Since λ is considered to be equal to $\beta(\lambda)$ (Section 2.1 above), $H(\lambda)$ is defined to be the identity matrix.

The range of the function H is generally restricted by the way in which the topological structure is implemented. For example, if a picture is described in only two dimensions (in the plane $z = 0$), every value of the function H must assume the following form:

$$\begin{bmatrix} h_{11} & h_{12} & 0 & 0 \\ h_{21} & h_{22} & 0 & 0 \\ 0 & 0 & h_{33} & 0 \\ h_{41} & h_{42} & 0 & h_{44} \end{bmatrix}$$

(This matrix is assumed to be non-singular, i.e. $h_{33} \neq 0$, $h_{44} \neq 0$, and $h_{22}h_{11} \neq h_{21}h_{12}$.) Since all elements which are not on the main diagonal and which are in the third column are zero, the transformation which this matrix represents maps every point in the $z = 0$ plane into a point in that plane. The fact that the off-diagonal elements in the third row are zeros insures that the transformed x and y coordinates do not depend on the value of z. The zeros in the last column insure that no perspective projection is performed. As a further example, a structure which describes a three-dimensional picture, but which does not describe rotation or perspective projection, requires that every value of H assume the following form, where none of the values h_{11}, h_{22}, h_{33}, or h_{44} is zero.

$$\begin{bmatrix} h_{11} & 0 & 0 & 0 \\ 0 & h_{22} & 0 & 0 \\ 0 & 0 & h_{33} & 0 \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

With these restrictions, the only transformations which can be performed are: (1) scaling of individual coordinates (via h_{11}, h_{22}, and h_{33}), (2) uniform scaling of all coordinates (via h_{44}), and (3) translation (via h_{41}, h_{42}, and h_{43}).

Values of H may be illustrated by considering a coordinate system to be associated with each entity. The coordinate system which is associated with an entity e is the coordinate system with respect to which $\beta(e)$ is drawn to produce e. H(e) represents the transformation which maps coordinates expressed with respect to this coordinate system into coordinates expressed with respect to the coordinate system for λ. For example, the coordinate systems which are associated with the entities represented in Figure 2.3 are shown in Figure 2.9. The values of the function H which yield the entities shown in this figure are the following:

$$H(e_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1+x_4 & y_1+y_4 & 0 & 1 \end{bmatrix}, \quad H(e_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ x_2+x_4 & y_2+y_4 & 0 & 1 \end{bmatrix},$$

$$H(e_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_3+x_4 & y_3+y_4 & 0 & 1 \end{bmatrix}, \quad H(e_4) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_4 & y_4 & 0 & 1 \end{bmatrix},$$

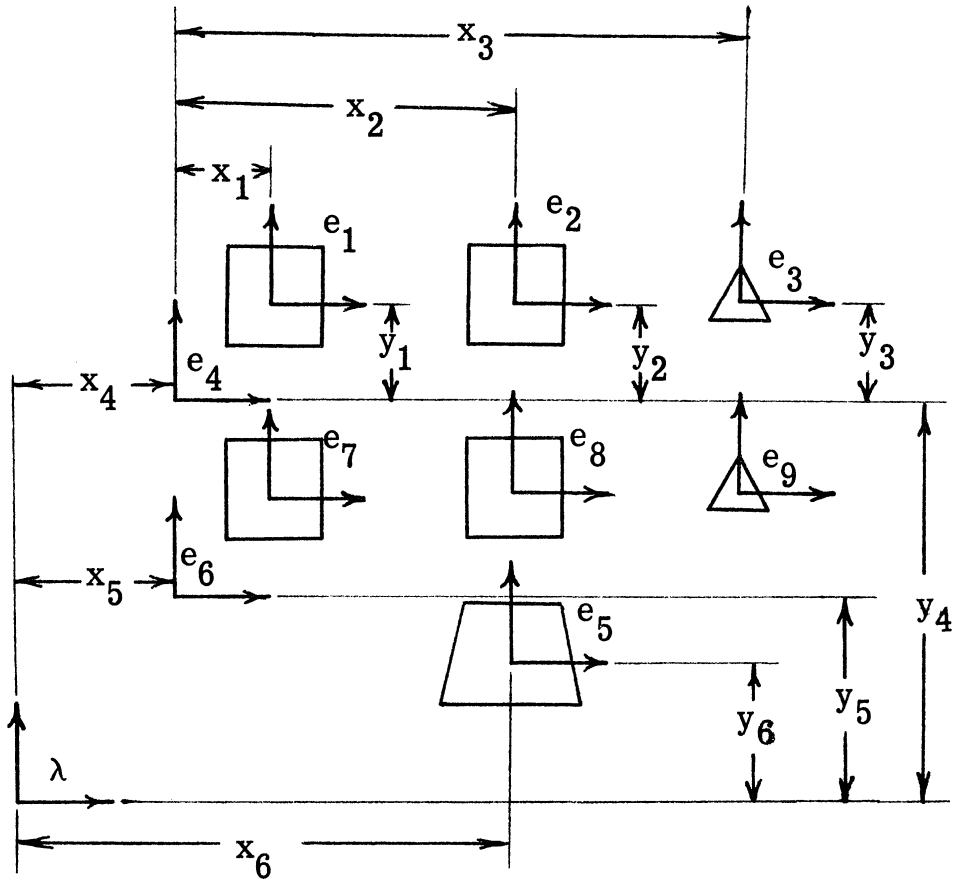


Figure 2.9

Coordinate Systems for Entities in Figure 2.3

$$H(e_5) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ x_6 & y_6 & 0 & 1 \end{bmatrix}, \qquad H(e_6) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_5 & y_5 & 0 & 1 \end{bmatrix},$$

$$H(e_7) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1+x_5 & y_1+y_5 & 0 & 1 \end{bmatrix}, \quad H(e_8) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_2+x_5 & y_2+y_5 & 0 & 1 \end{bmatrix},$$

and

$$H(e_9) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_3+x_5 & y_3+y_5 & 0 & 1 \end{bmatrix}.$$

To facilitate classification of those entities which may be represented as transformations of a common subpicture, the function F: A —> M is defined such that F(e,e') is a matrix which represents the transformation which maps the coordinates of points which define e' with respect to its associated coordinate system into coordinates expressed with respect to the coordinate system for e. H(e') may then be expressed as follows:

$$H(e') = F(e,e')H(e). \tag{2.15}$$

Since H(e) is non-singular, the following expression for F(e,e') is obtained by postmultiplying each side of this equation by $H(e)^{-1}$:

$$F(e,e') = H(e')H(e)^{-1}. \qquad (2.16)$$

Using this equation, the following values of F for the picture shown in Figure 2.9 may be calculated:

$$F(\lambda, e_4) = H(e_4) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ x_4 & y_4 & 0 & 1 \end{bmatrix},$$

$$F(\lambda, e_5) = H(e_5) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_6 & y_6 & 0 & 1 \end{bmatrix}, \quad \text{and}$$

$$F(\lambda, e_6) = H(e_6) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_5 & y_5 & 0 & 1 \end{bmatrix}.$$

Furthermore, since $\beta(e_4) = \beta(e_6)$,

$$F(e_4, e_1) = F(e_6, e_7) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & 0 & 1 \end{bmatrix},$$

$$F(e_4, e_2) = F(e_6, e_8) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_2 & y_2 & 0 & 1 \end{bmatrix}, \text{ and}$$

$$F(e_4, e_3) = F(e_6, e_4) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_3 & y_3 & 0 & 1 \end{bmatrix}.$$

2.1.2.3 Classification of Entities

As has been mentioned above, a relation $D \subset T$ is to be defined to describe which entities need not be stored independently. This relation is defined in terms of the functions β, F, and S as follows:

$$D = \bigcup_{i=0}^{m} D_i, \tag{2.17}$$

where

$$D_0 = \{(e, e') \mid (e, e') \in E_0 \times E_0 \text{ and } \beta(e) = \beta(e')\} \tag{2.18}$$

and

$$\begin{aligned} D_i = \{(e, e') \mid & (e, e') \in E_i \times E_i \text{ and there exists a} \\ & \text{one-one and onto function } \alpha\text{: } S(e) \longrightarrow S(e') \\ & \text{such that } e'' \in S(e) \Longrightarrow (e'', \alpha(e'')) \in \bigcup_{j=0}^{i-1} D_j \\ & \text{and } F(e, e'') = F(e', \alpha(e''))\}, \quad i=1, 2, \ldots, m. \end{aligned} \tag{2.19}$$

In defining this relation, the function β has been explicitly used only in the definition of D_0. Since all entities in E_0 have equivalent topologies, their relationship to other entities need not be considered in order to define D_0. However, this relationship must be considered in order to define D_i for $i=1, 2, \ldots, m$. Not only must the entities e and e' which are related by D_i have equivalent topologies and differ only by a transformation of coordinates, but each matrix in the image of $\{e\} \times S(e)$ under F must correspond to a matrix in the image of $\{e'\} \times S(e')$ under F as indicated.

Since the intended purpose of the relation D is to partition E into sets of entities which need not be represented independently, D must be an equivalence relation. In order to show that D is an equivalence relation, $D_0, D_1, D_2, \ldots, D_m$ are first shown to be equivalence relations by the following theorem:

Theorem 2.5

$D_0, D_1, D_2, \ldots, D_m$ are equivalence relations.

Proof

$D_0, D_1, D_2, \ldots, D_m$ must be shown to be reflexive, symmetric, and transitive.

1. $D_0, D_1, D_2, \ldots, D_m$ are shown to be reflexive by induction as follows:

Basis: $e \in E_0 \Longrightarrow \beta(e) = \beta(e) \Longrightarrow (e, e) \in D_0 \Longrightarrow D_0$ is reflexive.

Induction step: Assume that $D_0, D_1, D_2, \ldots, D_i$ are reflexive. Let $e \in E_{i+1}$ and $\alpha: S(e) \longrightarrow S(e)$ be the identity function. Then, $S(e) \subset \bigcup_{j=0}^{i} E_j$, and, since D_j is reflexive for $j \leq i$, $e' \in S(e) \Longrightarrow e' \in \bigcup_{j=0}^{i} E_j \Longrightarrow (e', e') \in \bigcup_{j=0}^{i} D_j \Longrightarrow (e', \alpha(e')) \in \bigcup_{j=0}^{i} D_j$. Furthermore, $F(e, e') = F(e, \alpha(e'))$. Consequently, $(e, e') \in D_{i+1}$ and D_{i+1} is reflexive.

2. $D_0, D_1, D_2, \ldots, D_m$ are shown to be symmetric by induction as follows:

Basis: $(e, e') \in D_0 \Longrightarrow \beta(e) = \beta(e') \Longrightarrow \beta(e') = \beta(e) \Longrightarrow (e', e) \in D_0$.

Induction step: Assume that $D_0, D_1, D_2, \ldots, D_i$ are symmetric. Let $(e, e') \in D_{i+1}$. Then, there exists a one-one and onto function $\alpha: S(e) \rightarrow S(e')$ such that $e'' \in S(e) \Longrightarrow (e'', \alpha(e'')) \in \bigcup_{j=0}^{i} D_j$ and $F(e, e'') = F(e', \alpha(e''))$. Since D_j is symmetric for $j \leq i$,

$e'' \in S(e) \Longrightarrow \alpha(e'') \in S(e') \Longrightarrow (\alpha(e''), e'') \in \bigcup_{j=0}^{i} D_j \Longrightarrow$

$(\alpha(e''), \alpha^{-1}(\alpha(e''))) \in \bigcup_{j=0}^{i} D_j$. Consequently, there exists the one-one and onto function α^{-1}: S(e')->S(e) such that $e''' \in S(e') \Longrightarrow$ $(e''', \alpha^{-1}(e''')) \in \bigcup_{j=0}^{i} D_j$ and $F(e', e''') = F(e, \alpha^{-1}(e'''))$. Then $(e', e) \in D_{i+1}$ and D_{i+1} is symmetric.

3. $D_0, D_1, D_2, \ldots, D_m$ are shown to be transitive by induction as follows:

Basis: $(e, e') \in D_0$ and $(e', e'') \in D_0 \Longrightarrow \beta(e) = \beta(e') = \beta(e'')$ $\Longrightarrow (e, e'') \in D_0$.

Induction step: Assume that $D_0, D_1, D_2, \ldots, D_i$ are transitive. Let $(e, e') \in D_{i+1}$ and $(e', e'') \in D_{i+1}$. Then, there exists a one-one and onto function α: S(e) $\longrightarrow$ S(e') such that $e''' \in S(e) \Longrightarrow$ $(e''', \alpha(e''')) \in \bigcup_{j=0}^{i} D_j$ and $F(e, e''') = F(e', \alpha(e'''))$, and there exists a one-one and onto function α': S(e') $\longrightarrow$ S(e'') such that $e''' \in S(e) \Longrightarrow$ $\alpha(e''') \in S(e') \Longrightarrow (\alpha(e'''), \alpha'(\alpha(e'''))) \in \bigcup_{j=0}^{i} D_j$ and $F(e', \alpha(e''')) =$ $F(e'', \alpha'(\alpha(e')))$. Since D_j is transitive for $j \leq i$, there exists the one-one and onto function $\alpha'\alpha$: S(e) $\longrightarrow$ S(e'') such that $e''' \in S(e) \Longrightarrow$ $(e''', \alpha'\alpha(e''')) \in \bigcup_{j=0}^{i} D_j$ and $F(e, e''') = F(e'', \alpha'\alpha(e'''))$. Consequently, $(e, e'') \in D_{i+1}$ and D_{i+1} is transitive. ∎

From the definition of D_i, $D_i \subset E_i \times E_i$ for i=0, 1, 2, . . . , m. Furthermore, as mentioned in Section 2. 3. 1 above, $E_i \cap E_j = \phi$ for $i \neq j$. Consequently, from Theorems 2. 4 and 2. 5, $D = \bigcup_{i=0} D_i$ is an equivalence relation.

D then defines the quotient set E/D, where each element of E/D is a set of entities which may be represented as transformations of a common subpicture. However, in the above discussion, only a static picture has been considered, i. e., the fact that an interactive display picture is continually being modified was ignored. In particular, reference has been made to values of F which are equal in in the definitions of $D_1, D_2, \ldots, D_m$. Whereas these definitions do yield a relation D which properly classifies entities for a static picture, they may not yield a relation D which allows the picture to be easily modified. For example, the picture shown in Figures 2. 1 and 2. 3 has been assumed to contain two rows (e_4 and e_6) of polygons which differ only in their positions. The definition of D given above may be applied to show that $\{e_4, e_6\} \in E/D$. However, if the square e_1 is moved, but the square e_7 is not moved, $F(e_4, e_1)$ no longer equals $F(e_6, e_7)$ and $\{e_4, e_6\} \notin E/D$. Consequently, the set E/D must be modified if the picture is modified in this way. However, if F(e, e') = F(e'', e''') is interpreted to mean "F(e, e') = F(e'', e''') as long as both e and e'' exist, " the necessity to modify E/D when a value of F is changed is avoided. For example, if the program is capable of moving square e_1 independently from square e_7, $\{e_4, e_6\}$

is never an element of E/D, since $F(e_4, e_1)$ is not considered to be equal to $F(e_6, e_7)$. Because of this practical consideration, equality of values of F should be interpreted in this way in the above definition of $D_1, D_2, \ldots, D_m$ and proof of Theorem 2.5, as well as in Step 5 of Algorithm 2.1 below.

2.1.3 Representation of Topology

In order to represent the entities in each equivalence class of D as transformations of a common subpicture, a set B of triples is defined to represent the instances which describe larger subpictures in terms of smaller subpictures. This set is defined as follows:

$$B = \{(D[e], D[e'], \eta(e, e')) \mid (e, e') \in A\}, \tag{2.20}$$

where $\eta : A \longrightarrow \{0, 1, 2, \ldots\}$ is a naming function for elements of A. Each element $(D[e], D[e'], \eta(e, e'))$ represents an instance of $\beta(e')$ which is used to form $\beta(e)$. The purpose of the function η is to distinguish different instances of $\beta(e')$ which are used to form $\beta(e)$, where $e' \in S(e)$. For example, the triples $(D[e_4], D[e_1], \eta(e_4, e_1))$ and $(D[e_4], D[e_2], \eta(e_4, e_2))$ in the picture represented by Figures 2.1 and 2.3 represent different instances of the square subpicture. These triples must be distinguished by the function η, since $\{e_1, e_2\} \subset E_0$ and $\beta(e_1) = \beta(e_2) \Longrightarrow D[e_1] = D[e_2]$. However, $(D[e_6], D[e_7], \eta(e_6, e_7))$ is to be the same as $(D[e_4], D[e_1], \eta(e_4, e_1))$, because $\beta(e_4) = \beta(e_6)$, $\beta(e_1) = \beta(e_7)$, and these two triples represent the same instance. For this to be true, $\eta(e_4, e_1)$ must be equal to

$\eta(e_6, e_7)$. Consequently, η does not associate a unique name with each element of A, and values of this function must be assigned systematically. A suitable algorithm for assigning values to this function is the following:

Algorithm 2.1

Procedure for assigning values to η, given A, F, and E/D.

(1) Let X = E/D.

(2) If $X = \phi$, terminate. Otherwise, select an equivalence class $D[e] \in E/D$ and remove this class from X. Select an element $e' \in D[e]$ and let $Y = D[e] - \{e'\}$. Assign an order to the elements of S(e'), and let $\eta(e', e_i) = i-1$, where e_i is the i-th element of S(e').

(3) If $Y = \phi$, proceed with Step 2. Otherwise, select an element $e'' \in Y$ and remove this element from Y. Let Z = S(e'').

(4) If $Z = \phi$, proceed with Step 3. Otherwise, select an element $e''' \in Z$ and remove this element from Z.

(5) Determine the element $e_i \in S(e')$ such that $F(e', e_i) = F(e'', e''')$ as long as both e' and e'' exist. Let $\eta(e'', e''') = i-1$ and proceed with Step 4. ∎

This algorithm assigns the same value of η to both $(e, e'') \in A$ and $(e', e''') \in A$ wherever $D[e] = D[e']$, $D[e''] = D[e''']$, and $F(e, e'') = F(e', e''')$ as long as both e and e' exist, whereas it assigns a different value of η to $(e, e'') \in A$ and $(e', e'') \in A$ wherever $D[e] = D[e']$

and either $D[e''] \neq D[e''']$ or $F(e, e'')$ may be modified to differ from $F(e', e''')$. The application of this algorithm to the picture described by Figures 2.1 and 2.3 is indicated below. The number of each step is shown, together with the values which are computed by performing it.

(1) $X = \{\{e_1, e_2, e_7, e_8\}, \{e_3, e_9\}, \{e_4, e_6\}, \{e_5\}, \{\lambda\}\}$

(2) $X = \{\{e_3, e_9\}, \{e_4, e_6\}, \{e_5\}, \{\lambda\}\}$

$e = e_1$, $Y = \{e_2, e_7, e_8\}$

$S(e_1) = \phi$. No values of η defined.

(3) $e' = e_2$, $Y = \{e_7, e_8\}$

$Z = \phi$

(3) $e' = e_7$, $Y = \{e_8\}$

$Z = \phi$

(3) $e' = e_8$, $Y = \phi$

$Z = \phi$

(2) $X = \{\{e_4, e_6\}, \{e_5\}, \{\lambda\}\}$

$e = e_3$, $Y = \{e_9\}$

$S(e_3) = \phi$. No values of η defined.

(3) $e' = e_9$, $Y = \phi$

$Z = \phi$

(2) $X = \{\{e_5\}, \{\lambda\}\}$

$e = e_4$, $Y = \{e_6\}$

$S(e_4) = \{e_1, e_2, e_3\}$. Let $\eta(e_4, e_1) = 0$, $\eta(e_4, e_2) = 1$, $\eta(e_4, e_3) = 2$.

(3) $e'' = e_6$, $Y = \phi$

$Z = \{e_7, e_8, e_9\}$

(4) $e''' = e_7$, $Z = \{e_8, e_9\}$

(5) $\eta(e_6, e_7) = 0$

(4) $e''' = e_8$, $Z = \{e_9\}$

(5) $\eta(e_6, e_8) = 1$

(4) $e = e_9$, $Z = \phi$

(5) $\eta(e_6, e_9) = 2$

(2) $X = \{\{\lambda\}\}$

$e = e_5$, $Y = \phi$

$S(e_5) = \phi$. No values of η defined.

(2) $X = \phi$

$e = \lambda$, $Y = \phi$

$S(\lambda) = \{e_4, e_5, e_6\}$. Let $\eta(\lambda, e_4) = 0$, $\eta(\lambda, e_5) = 1$, $\eta(\lambda, e_6) = 2$

The set B may be represented by a multigraph in which each element (D[e], D[e'], η(e, e')) is represented by a directed line from D[e] to D[e'] which is labeled η(e, e'). For example, this graph for the picture described by Figures 2. 1 and 2. 3 with the values of η assigned as shown above is shown in Figure 2. 10. This representation of B will be used in subsequent figures.

Each entity $e \neq \lambda$ may be identified by a set of sequences of elements of B which correspond to $(\lambda, e) \in C$. In order to simplify notation when describing this set, the function ρ: A$\rightarrow$ B is defined

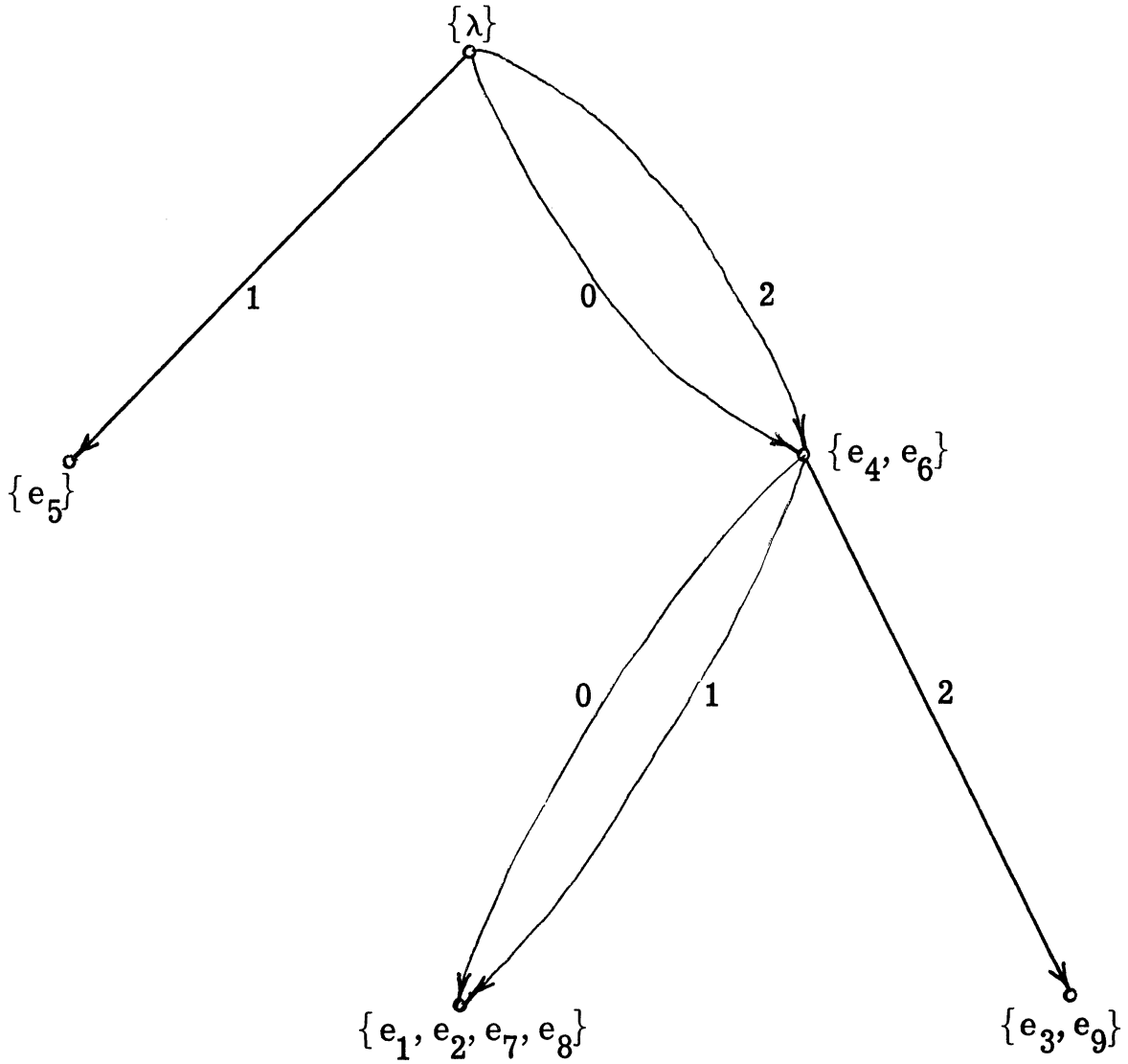


Figure 2. 10

Graphical Representation of B

such that

$$\rho(e, e') = (D[e], D[e'], \eta(e, e')). \tag{2.21}$$

Thus, $\rho(e, e')$ is an instance of $\beta(e')$ which is used to form $\beta(e)$. Note that, since ρ may map many elements of A into a single element of B, $\rho(e, e')$ does not contain sufficient information to determine e or e'. The following definition is then given to describe certain sequences of elements of B :

Definition 2. 5

A path of length n from entity e_1 to entity e_{n+1} is a sequence $(\rho(e_1, e_2), \rho(e_2, e_3), \ldots, \rho(e_n, e_{n+1})) \in B^n$.

Clearly, there exists at least one path from e to e' if and only if $(e, e') \in C$. However, if there exist several distinct entities $e_1, e_2, \ldots, e_n$ such that $e' \in \bigcap_{i=1}^{n} S(e_i)$, there may exist several paths from e to e'. For example, in the circuit diagram shown in Figure 2. 4, $e_{10} \in S(e_2) \cap S(e_3)$. There are two paths from λ to e_{10} : $(\rho(\lambda, e_2), \rho(e_2, e_{10}))$ and $(\rho(\lambda, e_3), \rho(e_3, e_{10}))$. Since there are no other paths from λ to e_{10}, the set of paths which identifies e_{10} is then the set of these two paths. Usually, however, every set of paths which identifies an entity consists of only one element. For example, the path which represents each entity in the picture which is represented by the set B in Figure 2. 10 is shown in Table 2. 1.

Entity	Path
e_1	$((\{\lambda\}, \{e_4, e_6\}, 0), (\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, 0))$
e_2	$((\{\lambda\}, \{e_4, e_6\}, 0), (\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, 1))$
e_3	$((\{\lambda\}, \{e_4, e_6\}, 0), (\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, 1))$
e_4	$((\{\lambda\}, \{e_4, e_6\}, 0))$
e_5	$((\{\lambda\}, \{e_5\}, 1))$
e_6	$((\{\lambda\}, \{e_4, e_6\}, 2))$
e_7	$((\{\lambda\}, \{e_4, e_6\}, 2), (\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, 1))$
e_8	$((\{\lambda\}, \{e_4, e_6\}, 2), (\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, 1))$
e_9	$((\{\lambda\}, \{e_4, e_6\}, 2), (\{e_4, e_6\}, \{e_3, e_4\}, 2))$

Table 2.1

Paths Which Represent Entities in Figure 2.10

2.2 Coordinate Transformations

In order that the generation of a picture from the topological structure be possible, there must exist a procedure for computing H(e) and β(e) for every e ϵ E. H(λ) is known to be the identity matrix and, therefore, need not be computed. For every class D[e] such that e ϵ E_0, β(e) must be specified explicitly as information external to the structure. However, the values H(e) for e ϵ E - E_m and β(e) for e ϵ E - E_0 must be computed from information in the structure and the image of E_0 under β.

For this computation to be possible, the structure must include both the set B and coordinate transformation matrices from which values of F may be computed. Then, by multiplying the values of F which are associated with the components of any path $(\rho(e_1, e_2), \rho(e_2, e_3), \ldots, \rho(e_n, e_{n+1}))$, the transformation matrix

$$\begin{aligned} &F(e_n, e_{n+1}) \ldots F(e_2, e_3)\, F(e_1, e_2) \\ &\quad = (H(e_{n+1})H(e_n)^{-1}) \ldots (H(e_3)H(e_2)^{-1})(H(e_2)H(e_1)^{-1}) \\ &\quad = H(e_{n+1})H(e_1)^{-1} \end{aligned} \tag{2.22}$$

is obtained. In particular, this transformation matrix for a path from λ to an entity e $\neq \lambda$ is H(e) $H(\lambda)^{-1}$ = H(e). The subpicture β(e) for e ϵ E - E_0 is produced by superimposing the images obtained by applying the transformation represented by H(e') $H(e)^{-1}$ to β(e') for

each entity e' such that there exists a path from e to e'. Since, for all entities e'' ϵ E - E_0. β(e'') is described as a superimposition of the images obtained by transforming other subpictures, the only images which are needed to generate β(e) are those which are used to generate entities in E_0. Since H(λ) represents the identity transformation, the picture λ is displayed by generating $\beta(\lambda)$, i.e., by superimposing the images obtained by applying H(e) to β(e) for each entity e ϵ E_0.

The representation of the relation A by the set B may be ambiguous in some cases if there are distinct entities e, e', and e'' such that e ϵ S(e') $\cap$ S(e''). For example, Figure 2. 11 depicts a relation A and the corresponding set B for which this ambiguity might occur. For this example, the fact that there are two paths from λ to e may generally be recognized by observing that the transformation matrices associated with the two paths ($\rho(\lambda, e')$, $\rho(e', e)$) and ($\rho(\lambda, e'')$, $\rho(e'', e)$) are equal, i.e. $F(e', e)\, F(\lambda, e') = F(e'', e)F(\lambda, e'') = H(e)$. However, if the entity e_1 coincides with e, the transformation matrix $F(\lambda, e_5)F(e_5, e_4)F(e_4, e_1)$ which is associated with the path $(\rho(\lambda, e_5), \rho(e_5, e_4), \rho(e_4, e_1))$ is also equal to H(e). In this event, this path may be interpreted to be a third path from λ to e, rather than a path from λ to e_1, when the set B and values of H are examined. D[e] then appears to contain only the one entity e ϵ $S(e_4)$ $\cap$ S(e') $\cap$ S(e''). Since individual entities in an equivalence class of D can differ only in their values of the function H, this ambiguity cannot be resolved in general. Care must be exercised when designing an

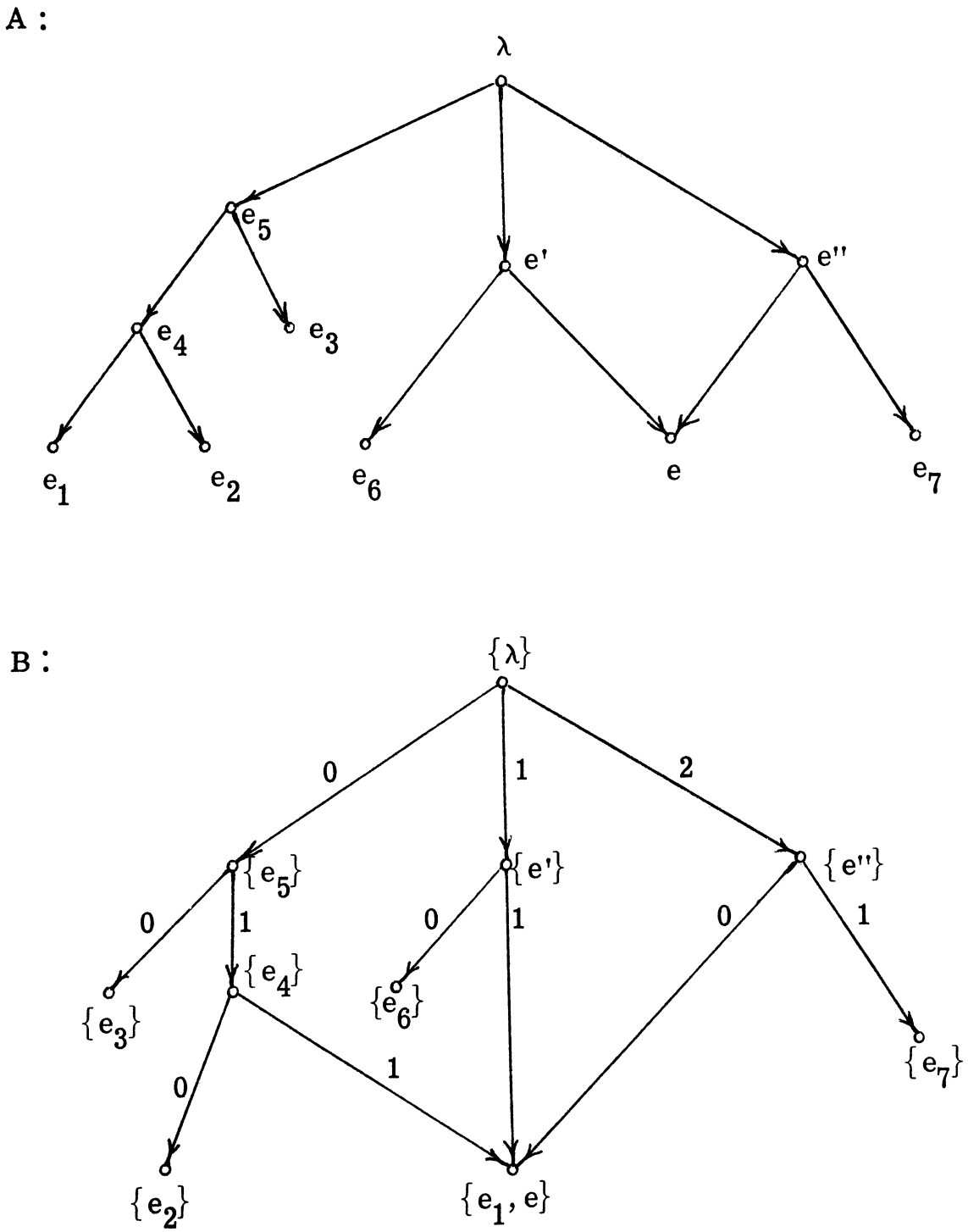


Figure 2. 11

An Ambiguous Topological Structure

application program, then, to insure that this ambiguity does not occur. For example, an application program which manipulates the structure shown in Figure 2. 11 might be constrained never to move an entity in an equivalence class of D so that it coincides with another member of the same class. If, however, the application requires that movement of entities not be constrained, a constraint which is consistent with the semantics of the application might be placed on the lengths of the paths from λ to all entities e such that $e \in S(e') \cap S(e'')$ for distinct entities e' and e''. For example, such paths might be required to have length two in Figure 2.11. The entity e_1 may then be recognized as being different from e because the path from λ to e_1 has length three.

2. 2. 1 A Structure Which Enforces No Constraints

As mentioned above, each coordinate transformation matrix H(e) is a product of values of F. Since all values of F which are associated with a single element of B are the same, all values of F which are needed to compute H are values of $F' : B \longrightarrow M$, where

$$F'(\rho(e, e')) = F(e, e'). \tag{2.23}$$

Thus, for any path $(\rho(\lambda, e_1), \rho(e_1, e_2), \ldots, \rho(e_n, e))$ from λ to $e \neq \lambda$, $H(e) = F'(\rho(e_n, e)) \ldots F'(\rho(e_1, e_2)) F'(\lambda, e_1))$. Since the picture λ may be produced by generating all entities in E_0, the set $\{B, E/D, F'\}$ represents a topological structure which, together with values of β for entities in E_0, describes the picture λ.

Whenever an entity e is modified in this structure, the only entities e' which are modified without some action by a constraint program are those for which either $(e', e) \in C$ or $(e, e') \in C$. For example, if the row e_4 in Figure 2.1 is moved, the picture λ is modified because $(\lambda, e_4) \in C$. Similarly, the squares e_1 and e_2 and the triangle e_3 are moved because $(e_4, e_1) \in C$, $(e_4, e_2) \in C$, and $(e_4, e_3) \in C$. The entities e_5, e_6, e_7, e_8 and e_9 are not modified, however, for they are not related to e_4 by C.

Various implementations of this topological structure have been widely used in previous interactive graphics programs, e.g. SKETCHPAD [67,68], SKETCHPAD III [28], GRAPHIC-2 [12], DIM [5], and PLAN-PGS [11]. This structure lends itself to independent modification of entities which are not related by C. However, as mentioned in Chapter 1, a constraint program is usually included as part of the interactive graphics program to refine the structure after it has been modified in response to a control language input. Some of the constraints which are enforced by this program cause entities which are not related by C to be modified in dependent fashion. For example, when a character is removed from a line of text which is displayed, other characters in the line may be adjusted so that no gap is left in the line. When a connection is drawn in a network diagram, network element symbols which are not related to this connection by C may be moved so that the diagram maintains an aesthetically pleasing appearance. In a picture of a

mechanical system, many parts may move when a specified part is moved in order to simulate mechanical constraints. If coordinate transformations are represented in the structure so that they depend on other parts of the structure, some constraints may be enforced as a consequence of refreshing the picture. One structure in which coordinate transformations are represented in this way is described below.

2. 2. 2 A Structure Which Enforces Concatenation Constraints

If coordinate transformations are represented so that they depend on portions of the structure, the subpictures which are used to generate entities in each set S(e) may be concatenated to form the subpicture β(e). Then, whenever one of these entities is modified, other entities are automatically modified in order to enforce the concatenation constraint. For concatenation of subpictures to have meaning, however, the entities in each set S(e) must be ordered.

2. 2. 2. 1 Ordering of Entities

If the picture is generated from the structure by a sequential process, the elements of S(e) for each entity $e \epsilon E$ are assigned an order for purposes of generating the picture. If this order can be controlled and examined by the program, it may aid the interpretation of the structure as well as permit subpictures to be concatenated. For example, this ordering is useful in conjunction with the network

shown in Figure 2. 4. In order to interpret a demonstrative input on this network, the program might need to determine whether the item referenced was an element symbol or a connection symbol. This is easily accomplished if an element symbol is known to be the first entity in every set S(e) such that e represents a circuit element, i. e., $e \in S(\lambda)$. Furthermore, although the capacitor and resistor elements are bilateral elements, an analysis program would need to determine which ports of the transistor correspond to the emitter, base, and collector of the transistor. This information could easily be represented in the structure by the order in which the ports of each transistor element are displayed.

In order to describe this ordering of entities, the functions $\delta: A \longrightarrow E$ and $\delta': A \longrightarrow E$ are defined as follows:

$$\delta(e, e') = \begin{cases} e', \text{ if } e' \text{ is the first entity in } S(e), \text{ or} \\ e'' \text{ such that } e'' \text{ immediately precedes } e' \text{ in } S(e), \\ \text{otherwise} \end{cases} \tag{2.24}$$

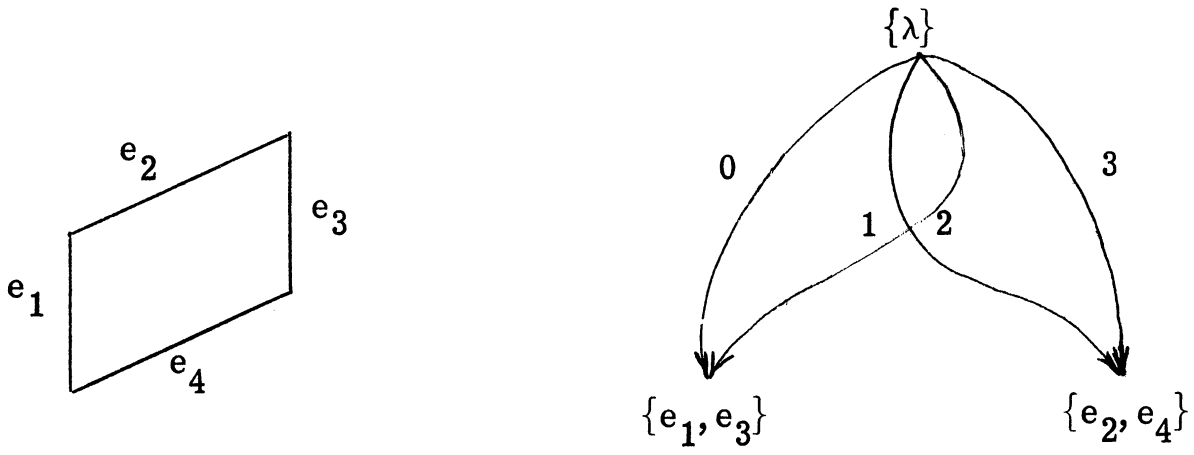
$$\delta'(e, e') = \begin{cases} e', \text{ if } e' \text{ is the last entity in } S(e), \text{ or} \\ e'' \text{ such that } e'' \text{ immediately follows } e' \text{ in } S(e), \\ \text{otherwise} \end{cases} \tag{2.25}$$

These functions are used below to describe the coordinate transformation matrices which are involved in concatenating subpictures.

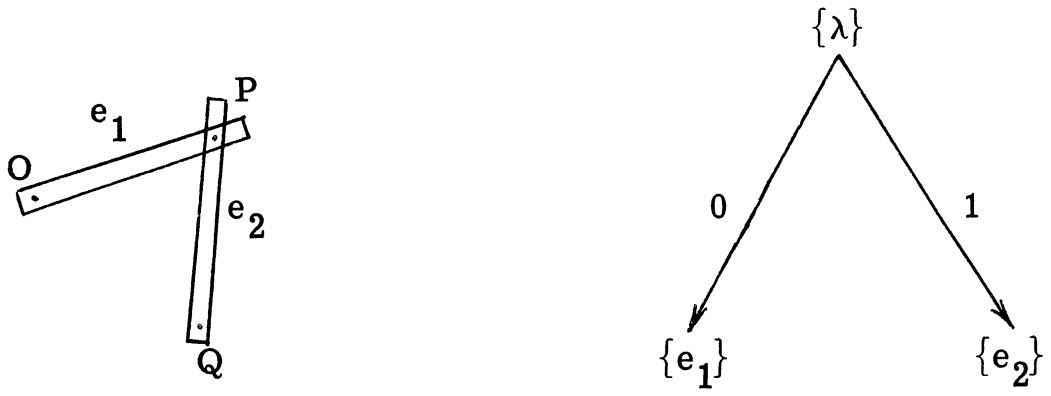
Before these matrices can be defined, however, concatenation of subpictures must be described.

2.2.2.2 Concantenation of Subpictures

Two simple examples of pictures for which the facility to concatenate subpictures is useful are shown in Figure 2.12. Each example is represented in this figure by a representative drawing and a diagram of the associated set B. The elements of B are arranged from left to right in each diagram in the order in which they are used to generate each picture. Figure 2.12a represents a parallelogram if the four lines which form the picture are constrained to be concatenated to each other at their end points and the only coordinate transformations which are represented in the structure are translations. Figure 2.12b represents an assembly of levers in which point Q is to be positioned by rotating lever e_1 about point O. If lever e_1 is rotated about point O and lever e_2 is to remain stationary with respect to lever e_1, lever e_2 must also rotate through the same angle about point P and must be translated so that it remains attached to lever e_1. The constraints which are needed for these examples may easily be enforced by representing concatenations of subpictures within the topological structure. Before a means of representing such concatenations is discussed, however, some relevant ideas developed by A. C. Shaw [61,62] are discussed.



a. A Parallelogram



b. An Assembly of Levers

Figure 2. 12

Examples of Pictures for Which a Concatenation Facility is Useful

Shaw was concerned with developing a picture parsing method which was well suited to the analysis, rather than the generation, of pictures, although his procedure could be applied to either process. Each subpicture was considered to have two distinguished points, called a head and a tail, at which it could be concatenated to other subpictures. Four binary operators were defined to describe the concatenation of subpictures at these points.

To facilitate discussion of the concatenation operators, each subpicture was represented abstractly by a directed line from its tail to its head. Each operator could then be represented graphically as a concatenation of two directed line segments. The four operators are represented in this way in Figure 2.13, and they are interpreted as follows:

$x + y$	Concatenate the head of x to the tail of y.
$x - y$	Concatenate the head of x to the head of y.
$x \times y$	Concatenate the tail of x to the tail of y.
$x * y$	Concatenate the head of x to the head of y and the tail of x to the tail of y.

In each case, the result which was formed by concatenating x and y was considered to have a tail which was the tail of x and a head which was the head of y. The result of each concatenation operation could then be used as an operand for further concatenation operations.

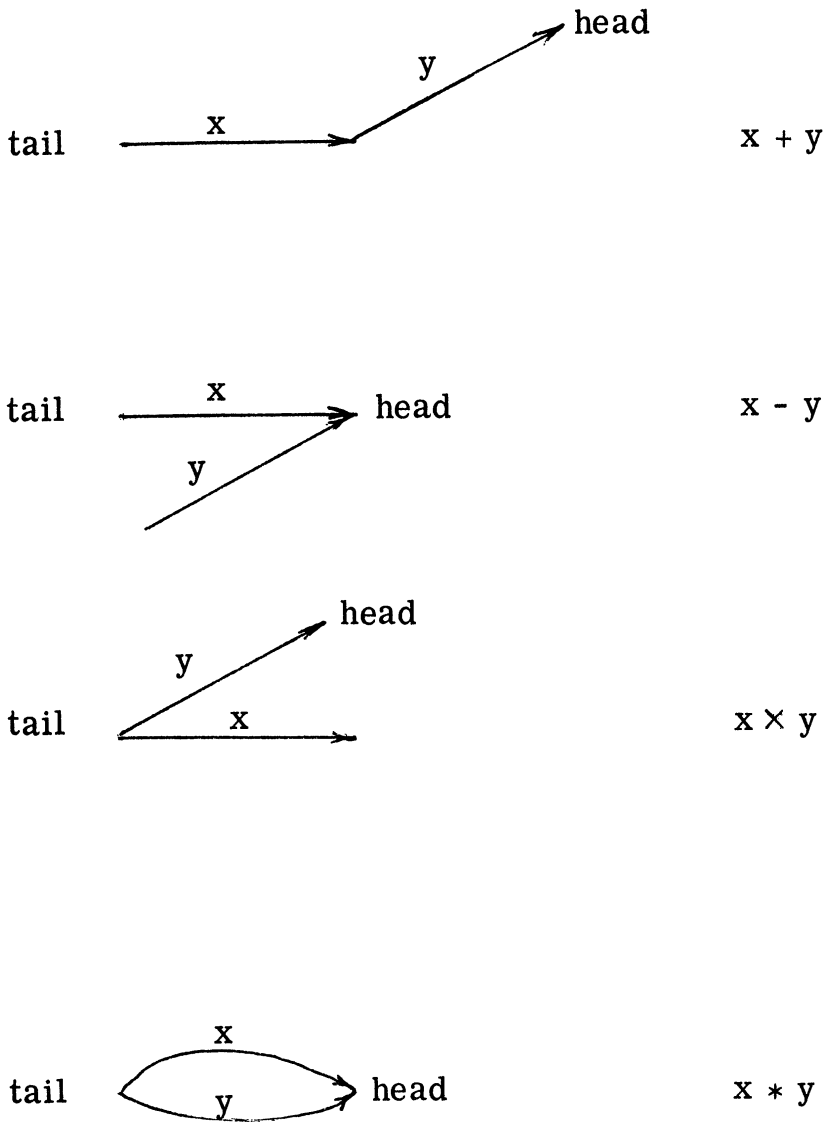


Figure 2.13

Shaw's Binary Concatenation Operators

2.2.2.2.1 Coordinate Systems for Concatenation

By using this basic concept of concatenation of subpictures, a topological structure which enforces concatenation constraints may be defined. Natural choices for the head and tail of each subpicture which is represented in this structure are the points at which the drawing of that subpicture is terminated and begun, respectively. However, at the time that an entity e is to be produced, not just the coordinates of a point are known, but the coordinate system with respect to which $\beta(e)$ is to be drawn in order to produce the entity is known. This coordinate system is the coordinate system which was associated with e in Section 2.1.2.2.2 above, and will be called the tail system of e when discussing concatenation of subpictures. $H(e)$ represents the coordinate transformation which maps coordinates expressed with respect to this coordinate system into coordinates expressed with respect to the tail system of λ. The process of drawing the subpicture $\beta(e)$ to produce e is considered to perform a coordinate transformation which maps coordinates expressed with respect to a second coordinate system into coordinates expressed with respect to the tail system of e. This second coordinate system will be called the head system of e.

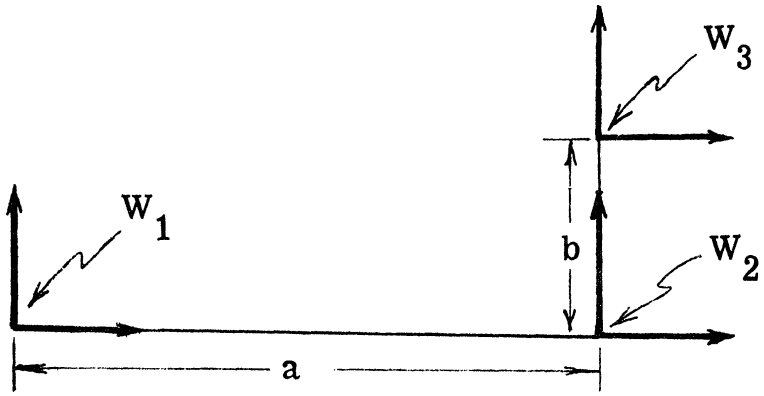
With these coordinate systems defined, concatenation operations similar to those defined by Shaw may be incorporated into the structure. However, since the relative positions of the head and tail of each operand

of Shaw's "*" operator must be equal, this operator cannot be applied to arbitrarily selected subpictures and will not be considered further.

2.2.2.2.2 Transformations Performed by Drawing a Subpicture

In order to describe these operations, a function G: E/D—>M is defined such that G(D[e]) is the matrix which represents the coordinate transformation which maps coordinates expressed with respect to the head system of each entity in D[e] into coordinates expressed with respect to its tail system. Thus, the matrix G(D[e])H(e) maps coordinates expressed with respect to the head system of e into coordinates expressed with respect to the tail system of λ. For simplicity, G(D[e])H(e) will be said to describe the head system of e with respect to the tail system of λ. Similarly, H(e) will be said to describe the tail system of e with respect to the tail system of λ, and G(D[e]) will be said to describe the head system of e with respect to its tail system.

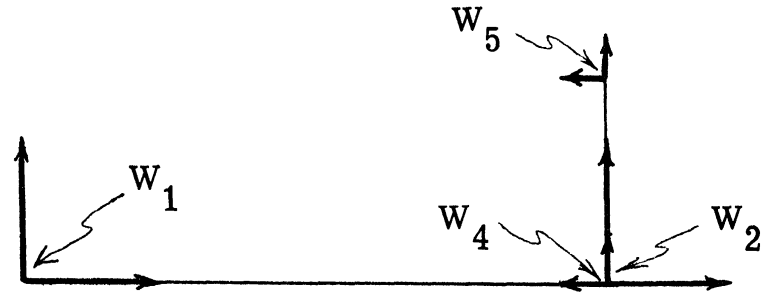
By associating coordinate transformations with each step of a procedure for drawing β(e), one way in which G(D[e]) may be determined by this procedure may be illustrated. Figure 2.14 depicts a two-dimensional subpicture β(e) which consists of two straight lines, together with two procedures for drawing this subpicture. For this example, the following two types of steps are applied to draw the subpicture:



Define W_2 by drawing vector (a,0) with respect to W_1.

Define W_3 by drawing vector (0,b) with respect to W_2.

$$G(D[e]) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & b & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & 0 & 1 \end{bmatrix}$$



Define W_2 by drawing vector (a,0) with respect to W_1.

Define W_4 by performing rotate-and-scale step with respect to W_2.

with $\alpha = \pi/2$ and $\sigma = a/b$.

Define W_5 by drawing vector (a,0) with respect to W_4.

$$G(D[e]) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & a/b \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a^2/b & a & 0 & a/b \end{bmatrix}$$

Figure 2.14

Example Transformation Matrices G(D[e])

(1) A vector with components x and y is drawn relative to a given coordinate system. The coordinate transformation which is associated with this step is described by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & 0 & 1 \end{bmatrix} .$$

(2) A rotate-and-scale step is performed relative to a given coordinate system in order to define a new coordinate system. This step contributes no visible parts to the subpicture. The coordinate transformation which is associated with this step is described by the matrix

$$\begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \sigma \end{bmatrix} .$$

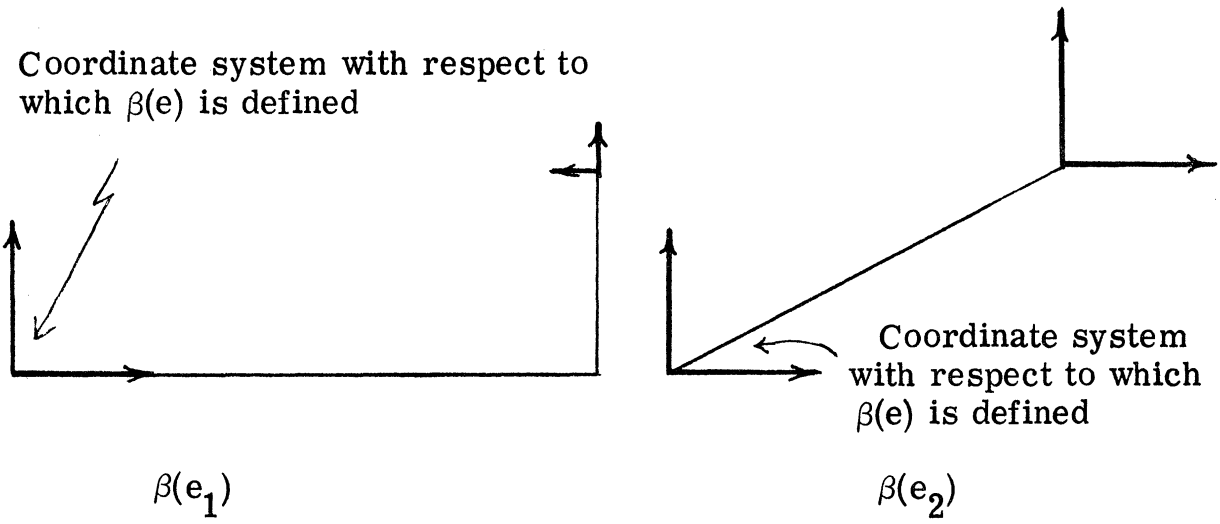
(In this matrix, α may be interpreted as an angle of rotation, whereas σ may be interpreted as a scale factor.)

In the first procedure indicated, a vector with components (a, 0) is drawn with respect to the coordinate system W_1. The coordinate transformation which is associated with this step describes the coordinate system W_2 with respect to W_1. A second vector with components (0, b) is then drawn with respect to W_2. The transformation which is associated with this step describes W_3 with respect to W_2. When this subpicture is drawn to produce an entity, the tail system of this entity is W_1, whereas the head system of this entity is W_3. In the second procedure indicated, two vectors with components (a, 0) are drawn. However, before the second vector is drawn, a rotate-and-scale step is performed to define the coordinate system W_4. The second vector is then drawn with respect to this coordinate system so that it becomes a vector with components (0, b) with respect to W_2. Note that each of these procedures yields a different matrix G(D[e]), although the subpicture which is produced by each procedure is otherwise the same.

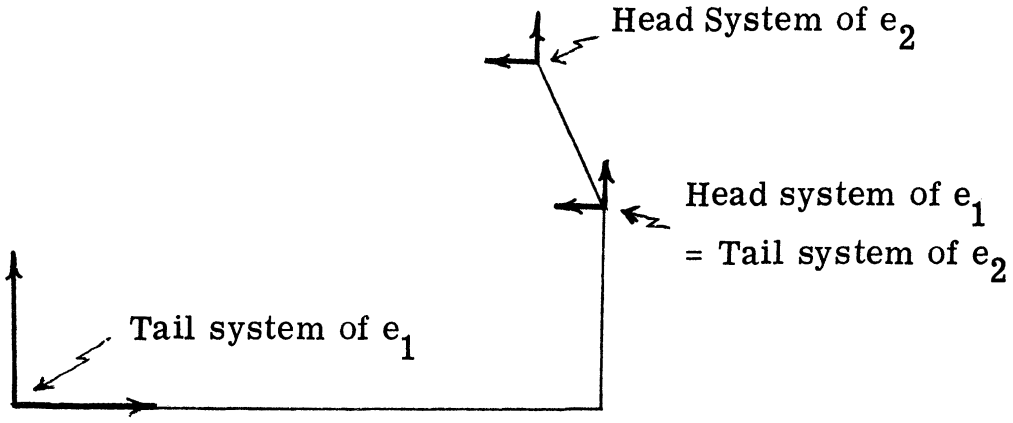
2.2.2.2.3 Concatenations as Coordinate Transformations

Two subpictures $\beta(e_1)$ and $\beta(e_2)$ may be concatenated as shown in Figure 2.15. In this figure, the concatenation is performed by considering the tail system of e_2 to be the head system of e_1, i.e., the coordinate system with respect to which $\beta(e_2)$ is drawn to produce e_2 is the head system of e_1. Because the head system of e_1 is defined by generating e_1, the concatenation operation shown in this figure is the one which is performed whenever e_1 and e_2 are generated in sequence. However, this is not the only concatenation operation which can easily be performed. These two subpictures may be concatenated in three other ways if a procedure is developed which allows either the head system or tail system of e_1 to be either the head system or tail system of e_2. These operations are shown in Figure 2.16. A method for performing these operations is described below.

In order to perform the concatenation operations shown in Figure 2.16, inverses of matrices which are values of G are considered. (Since G maps E/D into M, and M was defined to be a set of nonsingular matrices, these inverses always exist.) By selectively performing the coordinate transformations which are represented by $G(D[e_1])^{-1}$ and $G(D[e_2])^{-1}$ between the time that the generation of e_1 is completed and the generation of e_2 is begun, $\beta(e_1)$ and $\beta(e_2)$ may be concatenated as shown in either Figure 2.15 or 2.16.



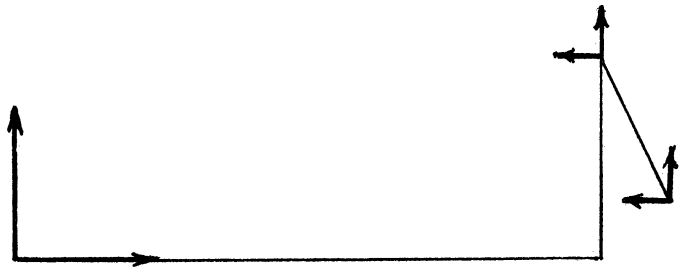
a. Subpictures to be Concatenated



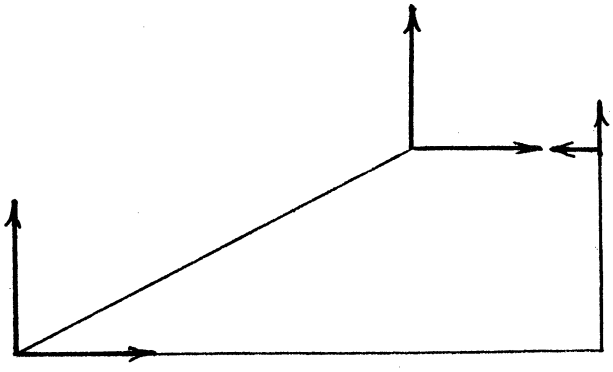
b. Concatenated Subpictures

Figure 2. 15

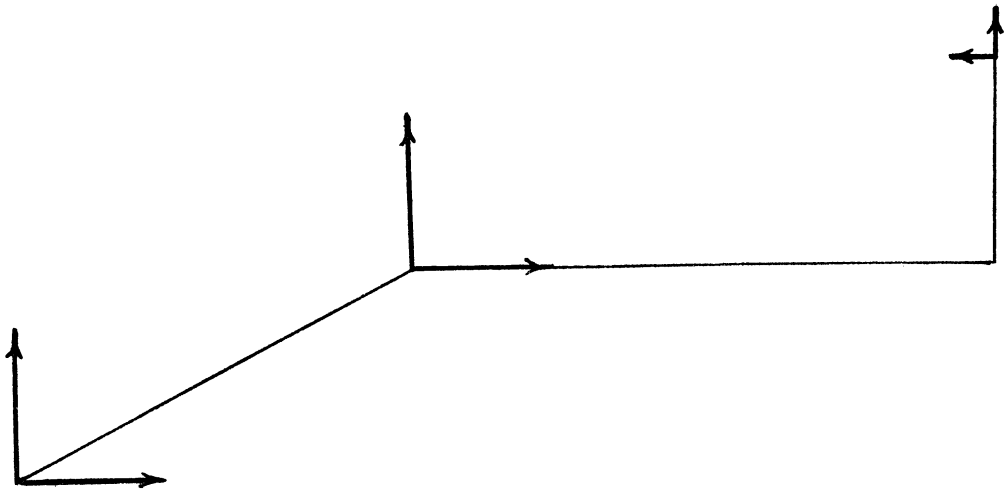
Concatenation of Two Subpictures



a. Head System of e_1 = Head System of e_2



b. Tail System of e_1 = Tail System of e_2



c. Tail System of e_1 = Head System of e_2

Figure 2. 16

Other Concatenations of Subpictures in Figure 2. 15

In order to describe the use of these matrices, the functions $G':B \longrightarrow M$ and $G'':B \longrightarrow M$ are defined such that $G'(\rho(e, e'))$ and $G''(\rho(e, e'))$ may each assume only the values I and $G(D[e'])^{-1}$. If $\delta(e, e') \not\in e'$ and $G'(\rho(e, e')) = I$, the tail system of e' is selected to be either the head system or tail system of $\delta(e, e')$, whereas, if $G'(\rho(e, e')) = G(D[e'])^{-1}$, the head system of e' is selected. Similarly, if $\delta'(e, e') \not\in e'$ and $G''(\rho(e, e')) = I$, the head system of e' is selected to be either the head system or tail system of $\delta'(e, e')$, whereas, if $G''(\rho(e, e')) = G(D[e'])^{-1}$, the tail system of e' is selected. Thus, values of G' and G'' specify the uses of the head systems and tail systems of the various entities.

The interpretation of G' and G'' may be illustrated by considering Figures 2. 15 and 2. 16. In these figures, all possible concatenations of $\beta(e_1)$ and $\beta(e_2)$ are shown. The values of G' and G'' which specify these concatenations are the following, where it is assumed that $\{e_1, e_2\} \subset S(e)$ and $\delta(e, e_2) = e_1$:

	Figure 2. 15	Figure 2. 16a	Figure 2. 16b	Figure 2. 16c
$G''(e_1)$	I	I	$G(D[e_1])^{-1}$	$G(D[e_1])^{-1}$
$G'(e_2)$	I	$G(D[e_2])^{-1}$	I	$G(D[e_2])^{-1}$

The values of G' and G'' have been chosen to be coordinate transformation matrices so that the selection of head systems and tail systems may be described as transformations of coordinates. In order to discuss this interpretation, the function $\gamma:B \longrightarrow M^3$ is defined as follows :

$$\gamma(\rho(e, e')) = (G'(\rho(e, e')),\ G(D[e']),\ G''(\rho(e, e'))). \qquad (2.26)$$

Each sequence $\gamma(\rho(e, e'))$ represents the sequence of coordinate transformations to be performed whenever an element of $D[e']$ is generated as a part of an element of $D[e]$. The transformation which is represented by $G(D[e'])$ is performed when $\beta(e')$ is drawn with respect to the tail system of an entity in $D[e']$. The transformation matrices $G'(\rho(e, e'))$ and $G''(\rho(e, e'))$, however, either are computed by some other means or are stored explicitly in the structure.

To interpret the sequence $\gamma(\rho(e,e'))$, consider an entity e' which is about to be produced as a part of e, and assume that $\delta(e,e') \neq e'$ and $\delta'(e,e') \neq e'$. Depending on the value of $G''(\rho(e, \delta(e,e')))$, the last coordinate system which was defined is either the tail system or the head system of $\delta(e,e')$. The tail system of e' is then that coordinate system which is described relative to this coordinate system by $G'(\rho(e,e'))$. If $G'(\rho(e,e')) = I$, the tail system of e' is clearly either the head system or tail system of $\delta(e,e')$. However, if $G'(\rho(e,e')) = G(D[e'])^{-1}$, the tail system of e' is generally a different coordinate system. Now consider the head system of e'. The process of drawing $\beta(e')$ describes the head system of e' with respect to the tail system of e'. Consequently, the head system of e' is described with respect to the head system or tail system of $\delta(e, e')$ by the matrix $G(D[e'])\ G'(\rho(e, e'))$.

Thus, if $G'(\rho(e, e')) = G(D[e'])^{-1}$, $G(D[e'])\, G'(\rho(e, e')) = I$, and the head system of e' is either the head system or tail system of $\delta(e, e')$. Either the head system or tail system of e' is then selected to be the coordinate system which is passed on to $\delta'(e, e')$. If $G''(\rho(e, e')) = I$, the head system of e' is selected, since it is the last coordinate system which was computed. However, if $G''(\rho(e, e')) = G(D[e'])^{-1}$, the tail system of e' is selected.

In the above discussion, $G'(\rho(e, e'))$ has been interpreted only for $\delta(e, e') \neq e'$, and $G''(\rho(e, e'))$ has been interpreted only for $\delta'(e, e') \neq e'$. Each of these matrices has a similar interpretation for $\delta(e, e') = e'$ or $\delta'(e, e') = e'$. If $\delta(e, e') = e'$, the tail system of e, rather than either the head system or tail system of $\delta(e, e')$, is selected to be the head system or tail system of e' according to $G'(\rho(e, e'))$. Similarly, if $\delta'(e, e') = e'$, the head system of e is selected to be the head system or tail system of e' according to $G''(\rho(e, e'))$. The matrix $G(D[e])$ for $e \in E - E_0$ is then the product of matrices which are components of sequences $\gamma(\rho(e, e'))$. In particular, if $S(e) = \{e_1, e_2, \ldots, e_n\}$, $\delta(e, e_1) = e_1$, and $\delta(e, e_{i+1}) = e_i$ for $i = 1, 2, \ldots, n - 1$,

$$G(D[e]) = G''(\rho(e, e_n))G(D[e_n])G'(\rho(e, e_n)) \ldots$$
$$G''(\rho(e, e_2))G(D[e_2])G'(\rho(e, e_2))G''(\rho(e, e_1))G(D[e_1])G'(\rho(e, e_1)). \quad (2.27)$$

2.2.2.3 Values of F Determined by Concatenation

With the functions G' and G'' defined, the matrix F(e, e') may be computed for each (e, e') ϵ A as follows:

$$F(e, e') = \begin{cases} G'(\rho(e, e')), \text{ if } \delta(e, e') = e', \text{ or} \\ \\ G'(\rho(e, e'))G''(\rho(e, \delta(e, e')))G(D[\delta(e, e')]F(e, \delta(e, e')), \\ \text{otherwise.} \end{cases} \tag{2.28}$$

In order to interpret this equation, the process of drawing $\beta(e)$ to produce e is considered. This process begins with the tail system of e computed. The tail system of e_1 is then described with respect to this tail system by $G'(\rho(e, e_1))$, where e_1 is the first entity in S(e). Since $G'(\rho(e, e_1))$ describes the tail system of e_1 with respect to the tail system of e, $F(e, e_1) = G'(\rho(e, e_1))$. To compute $F(e, e_2)$, where $\delta(e, e_2) = e_1$, the other transformations which are performed before the generation of e_2 is begun must be considered. These transformations are represented by the matrices $G(D[e_1])$ and $G''(\rho(e, e_1))$, respectively. Furthermore, the additional transformation which is represented by $G'(\rho(e, e_2))$ must be performed to produce the tail system of e_2. The tail system of e_2 is then described with respect to the tail system of e_1 by the transformation matrix $G'(\rho(e, e_2))\ G''(\rho(e, e_1))\ G(D[e_1])$, i.e., $F(e, e_2) = G'(\rho(e, e_2))\ G''(\rho(e, e_1))\ G(D[e_1])\ F(e, e_1)$. The matrices $F(e, e_i)$ are computed in similar manner for other entities $e_i \epsilon$ S(e).

2.2.2.4 Description of the Structure

Since values of F may be computed in this way, a topological structure which represents concatenations of subpictures may be described. As indicated above, the functions γ and δ are needed to compute F. However, if the topology of the picture is to be represented by the set B, the set E and the relation A are not explicitly represented, and the function $\delta: A \longrightarrow E$ is not useful. For this reason, the function $\mu: B \longrightarrow B \cup E/D$ is defined as follows:

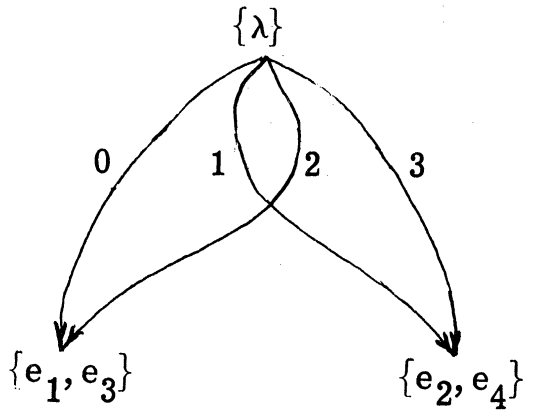
$$\mu(\rho(e, e')) = \begin{cases} D[e], \text{ if } \delta(e, e') = e', \text{ or} \\ \\ \rho(e, \delta(e, e')), \text{ otherwise.} \end{cases} \qquad (2.29)$$

All information which is necessary to compute F may be obtained from the functions μ and γ. If $\delta(e, e') = e'$, the first component $G'(\rho(e, e'))$ of $\gamma(\rho(e, e'))$ is the matrix $F(e, e')$. The fact that $\delta(e, e') = e'$ is detected by noting that $\mu(\rho(e, e'))$ belongs to E/D, rather than B. If $\delta(e, e') \neq e'$, components of both $\gamma(\rho(e, e'))$ and $\gamma(\mu(\rho(e, e')))$, as well as the matrix $F(e, \delta(e, e'))$, are used to compute $F(e, e')$. (If $F(e, e')$ is computed after $F(e, \delta(e, e'))$ is computed, $F(e, \delta(e, e'))$ is known when it is needed.) A topological structure which represents concatenation of subpictures is then described by the set $\{B, E/D, \mu, \gamma\}$.

2.2.2.5 Examples of the Structure

Figure 2.17 shows a structure of this form for each of the example pictures in Figure 2.12. In this figure, the values of γ are given explicitly, and the function μ is represented by assigning values of η such that $\eta(e, e') = \eta(e, e'') + 1$ wherever $\mu(\rho(e, e'')) = \rho(e, e')$. The structure shown in Figure 2.17a guarantees that the picture is a parallelogram as long as the subpictures $\beta(e_1) = \beta(e_3)$ and $\beta(e_2) = \beta(e_4)$ are each straight lines which may be concatenated at their end points. Since the only coordinate transformations which are represented in this structure are translations, all entities which are formed from one of these subpictures must be parallel lines of equal length. The only additional requirement needed to insure that the figure is a parallelogram, then, is that $G(\{\lambda\})$ be the identity matrix, i.e., that the figure be a closed polygon. Clearly, $G(\{\lambda\}) = G(\{e_2, e_4\})^{-1}G(\{e_1,e_3\})^{-1}G(\{e_2, e_4\})G(\{e_1, e_3\})$. Since both $G(\{e_1, e_3\})^{-1}$ and $G(\{e_2, e_4\})$ represent translations, the transformations which they represent are commutative, and $G(\{\lambda\}) = G(\{e_2, e_4\})^{-1}G(\{e_2, e_4\})G(\{e_1, e_3\})^{-1}G(\{e_1, e_3\}) = I$.

For purposes of discussion, assume that the levers in the system represented in Figure 2.17b are to be modified by modifying the subpictures $\beta(e_1)$ and $\beta(e_2)$. The structure insures that the tail system of e_2 is equal to the head system of e_1. Consequently, if e_1 is rotated by replacing $\beta(e_1)$ with a rotated form of $\beta(e_1)$, i.e., if $\beta(e_1)$ is modified so that $G(D[e_1])$ becomes



$$G(\{e_1, e_3\}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & 0 & 1 \end{bmatrix}$$

$$G(\{e_2, e_4\}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_2 & y_2 & 0 & 1 \end{bmatrix}$$

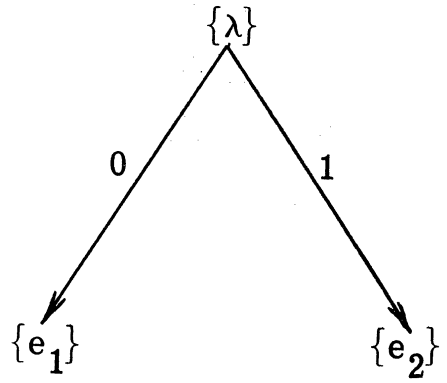
$$\gamma(\rho(\lambda, e_1)) = (I, G(\{e_1, e_3\}), I)$$

$$\gamma(\rho(\lambda, e_2)) = (I, G(\{e_2, e_4\}), I)$$

$$\gamma(\rho(\lambda, e_3)) = (G(\{e_1, e_3\})^{-1}, G(\{e_1, e_3\}), G(\{e_1, e_3\})^{-1})$$

$$\gamma(\rho(\lambda, e_4)) = (G(\{e_2, e_4\})^{-1}, G(\{e_2, e_4\}), G(\{e_2, e_4\})^{-1})$$

a. Parallelogram in Figure 2.12a



$$G(\{e_1\}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & 0 & 1 \end{bmatrix}$$

$$G(\{e_2\}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_2 & y_2 & 0 & 1 \end{bmatrix}$$

$$\gamma(\rho(\lambda, e_1)) = (I, G(\{e_1\}), I)$$

$$\gamma(\rho(\lambda, e_2)) = (I, G(\{e_2\}), I)$$

b. Lever System in Figure 2.12b

Figure 2.17

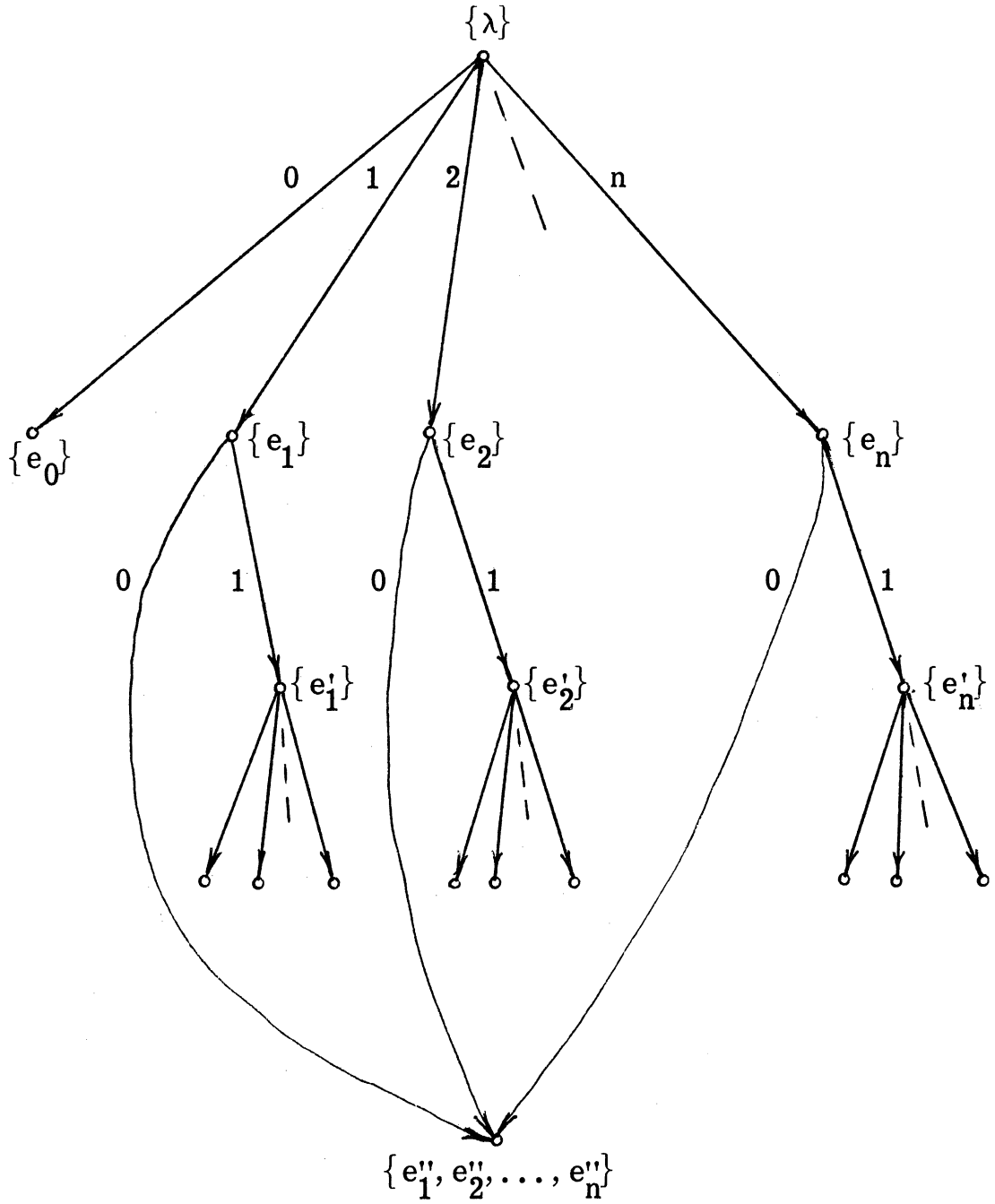
Applications of the Concatenation Facility

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where α is the angle of rotation, e_2 is translated so that it remains concatenated to e_1, and it is rotated through the angle α. However, if $\beta(e_2)$ is replaced with a rotated form of $\beta(e_2)$, e_2 is rotated about the origin of its tail system, and e_1 is not modified.

In practice, topological structures which represent concatenations of subpictures are more complex than those shown in Figure 2.17. For example, consider a program which provides for the editing of text via an interactive computer display. A page of text is to be displayed on a two-dimensional display so that the user can insert and delete both characters and lines. A topological structure which represents concatenation is useful here because, as an item is inserted, the text must be adjusted to provide room for the insertion. Furthermore, as text is deleted, the remaining text must be adjusted to close the gaps which are created by this process.

A suitable form of the topological structure for use with this program is indicated in Figure 2.18. The picture is composed of e_0 and the n lines of text $e_1, e_2, \ldots, e_n$. The entity e_0 is an invisible entity whose purpose is to establish a tail system for e_1 which is different from the coordinate system with respect to which the picture is drawn. For example, the picture may be drawn with respect to a coordinate system whose origin is at the center



$$\gamma(\rho(e_i, e_i')) = (I, G(\{e_i'\}), G(\{e_i'\})^{-1})$$

$$\gamma(\rho(e, e')) = (I, G(D[e']), I)$$

for $(e, e') \neq (e_i, e_i')$, $i=1, 2, \ldots, n$

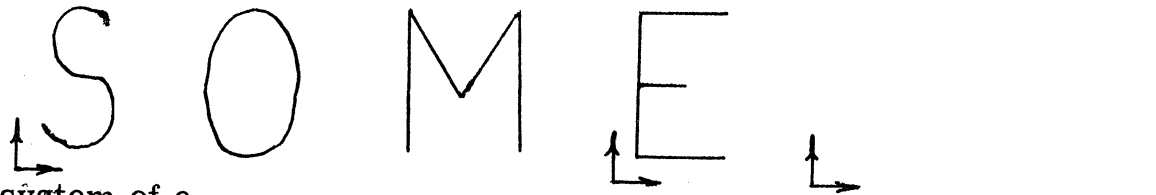
Figure 2.18

A Topological Structure for an Editor Program

of the screen. However, it is usually desirable to begin a page of text in the upper left hand corner of the screen. Each line e_i, $i=1, 2, \ldots, n$, is composed of a vertical displacement e''_i and a string of characters e'_i. Each line is considered to have these two parts so that the position of subsequent lines will not be affected by the number of characters which it contains. This independence of the number of characters in a line is achieved by ignoring the head system of each character set e'_i, i. e., by choosing $\gamma(\rho(e_i, e'_i))$ to be $(I, G(\{e'_i\}), G(\{e'_i\})^{-1})$.

The concatenations which are involved in describing a page of text are illustrated in Figure 2.19. Each character has a tail system at its lower left hand corner and a head system at its lower right hand corner, as illustrated for the character "E" in the word "SOME" in the figure. Several of the character subpictures which are concatenated to each other form a character string which has a tail system at its lower left hand corner and a head system at its lower right hand corner. An example of such a string is the string "SOME" which is denoted e'_1 in Figure 2.18. A line of text is formed by concatenating the subpicture for a text string to an invisible subpicture so that the head system of the line is the tail system of the text string and the tail system of the line is positioned above this head system. In Figure 2.19, the line e_1 is formed by concatenating the invisible subpicture $\beta(e''_1)$ to the subpicture $\beta(e'_1)$ in this way. Finally, the page of text is formed by concatenating

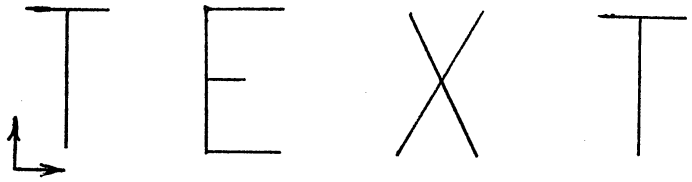
Tail system of e_1
= Tail system of e_1''



Head system of e_1
= Tail system of e_2
= Tail system of e_1'
= Head system of e_1''

Tail system of "E"
= Head system of "M"

Head system of "E"
= Head system of e_1'



Head system of e_2

Figure 2. 19

Concatenations of Text Characters

subpictures for the lines of text.

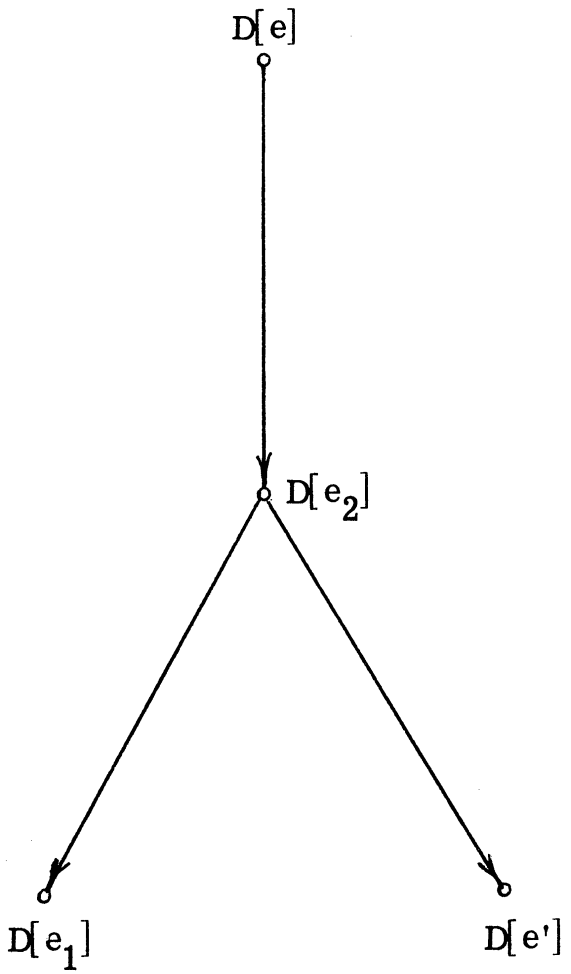
The purpose of this structure is to facilitate the insertion and deletion of text. The character e may be inserted into the line e_i after the character e' by creating the element $\rho(e_i', e) \in B$ such that $\mu(\rho(e_i', e)) = \rho(e_i', e')$. If this is done, all characters to the right of e' in line e_i are shifted right one character width to provide room for the new character. Similarly, a character e may be deleted by destroying the element $\rho(e_i', e)$. If this is done, all characters to the right of e in the line e_i are shifted left one character width to close the gap in the line. A line of text may be inserted below the line e_i on the page by inserting it after line e_i in the structure. All lines which are below e_i are then moved down to provide room for the new line. Similarly, if a line is deleted, all lines which were below it are moved up on the page to close the gap left between lines.

2. 2. 3 Applicability of Structures

Although the structure $\{B, E/D, \mu, \gamma\}$ provides for concatenation of subpictures, it does exhibit some disadvantages when compared to the structure $\{B, E/D, F'\}$. In particular, invisible entities are often needed to introduce coordinate transformations into the structure. Examples of these invisible entities are the entities e_0 and $e_1'', e_2'', \ldots, e_n''$ in Figure 2.18. The purpose of e_0 is to translate the origin of the page to the upper left hand corner of the screen, whereas the purpose of each entity e_i'', i=1, 2, . . . , n, is to translate

each line of text so that it is displayed below the previous line. Additional entities to those which are necessary to define the picture may also be needed to suppress the effect of certain values of the function G. For example, the entities $e_1', e_2', \ldots, e_n'$ in Figure 2.18 serve this purpose. Since $\gamma(\rho(e_i, e_i')) = (I, G(D[e_i']), G(D[e_i'])^{-1})$ for $i=1, 2, \ldots, n$, $G(D[e_i'])$ does not affect the position of any entities in the picture. Each of these entities which contributes only coordinate transformation information to the structure and which is not otherwise needed to define the picture will be termed an artificial entity.

The artificial entities which may be needed in a structure $\{B, E/D, \mu, \gamma\}$ in order to represent a coordinate transformation in the corresponding structure $\{B, E/D, F'\}$ are depicted in Figure 2.20. This figure shows the representation of $\rho(e, e')$ which is needed for each entity which is to be modified independently of those other entities which are not related to it by C. The entities in $D[e_1]$ are invisible entities which introduce $F(e, e')$ into the structure, whereas the entities in $D[e_2]$ are needed to suppress the effect of $G(D[e'])$. If the structure shown in Figure 2.20 requires more storage or time to interpret than the element $\rho(e, e') \in B$ and the matrix $F'(\rho(e, e')$ in the structure $\{B, E/D, F'\}$, the structure $\{B, E/D, F'\}$ may be more suitable than the structure $\{B, E/D, \mu, \gamma\}$ for those applications in which entities are not frequently concatenated.



$$G(D[e_1]) = F(e, e')$$

$$G'(\rho(e_2, e')) = I$$

$$\gamma(\rho(e, e_2)) = (I, G(D[e_2]), G(D[e_2])^{-1})$$

$$\gamma(\rho(e_2, e_1)) = (I, G(D[e_1]), I)$$

Figure 2.20

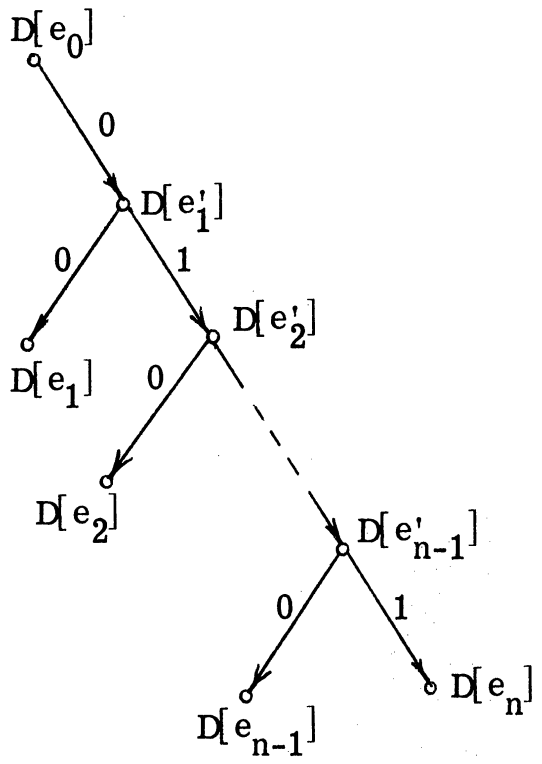
Isolation of Coordinate Transformations in the Structure $\{B, E/D, \mu, \gamma\}$

Although the structure $\{B, E/D, \mu, \gamma\}$ is inferior to the structure $\{B, E/D, F'\}$ for some applications, it is often more suitable for those applications which involve frequent concatenations of subpictures. Concatenation of subpictures can be represented by the structure $\{B, E/D, F'\}$. However, this type of representation requires that some artificial entities be defined, and it requires more computation to modify. These two methods are compared in Figure 2.21. In this figure, the subpicture $\beta(e_0)$ is represented as a concatenation of the subpictures $\beta(e_1), \beta(e_2), \ldots, \beta(e_n)$. 2n elements of E/D and 2n-1 elements of B are required to represent $\beta(e_0)$ in this way if the structure is described $\{B, E/D, F'\}$ whereas only n+1 elements of E/D and n elements of B are required if the structure is described by $\{B, E/D, \mu, \gamma\}$. Furthermore, if one of the subpictures $\beta(e_i)$, $i=1, 2, \ldots, n$, is modified such that $G(D[e_i])$ is changed, new transformation matrices $F(e_0, e_i)F(e_0, e_{i-1})^{-1}$ and $F(e_0, e_{i+1})F(e_0, e_i)^{-1}$ must be computed if the former structure is used, whereas no new matrix need be computed if the latter structure is used. These considerations justify the use of the structure $\{B, E/D, \mu, \gamma\}$ for some applications.

2.3 Conclusion

Two structures which represent the topology of a picture have been described. In either structure, each entity is represented by a coordinate transformation and the subpicture to which this transformation is to be applied to produce the entity. By considering entities,

$\{B, E/D, F'\}$:



$F'(\rho(e_0, e_1')) = G'(\rho(e_0, e_1))$

$F'(\rho(e_i', e_{i+1}')) = G'(\rho(e_0, e_{i+1}))G''(\rho(e_0, e_i))G(D[e_i]),\ i=1, 2, \ldots, n-2$

$F'(e_i', e_i) = I,\quad i=1, 2, \ldots, n-1$

$F'(\rho(e_{n-1}', e_n)) = G'(\rho(e_0, e_n))G''(\rho(e_0, e_{n-1}))G(D[e_n])$

$\{B, E/D, \mu, \gamma\}$:

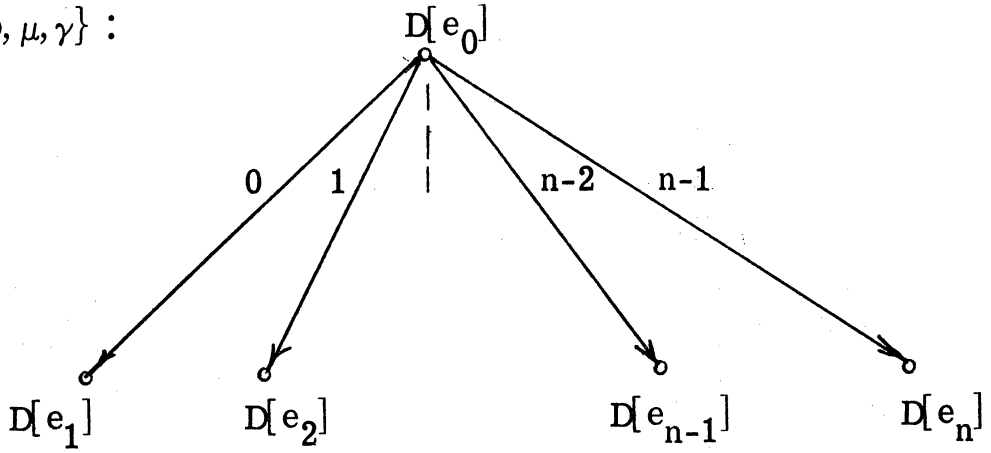


Figure 2.21

Alternative Representations of Concatenation

rather than subpictures, as a basis for defining these structures, a method of representing multiple uses of a particular picture part, as well as multiple occurrences of a subpicture, has been described. The two structures differ in that coordinate transformations are represented in one to facilitate independent modification of entities which are not topologically related, whereas these transformations are represented in the other to enforce concatenation constraints. These structures will be used as a basis for comparing various implementations of topological structures in succeeding chapters.

Chapter 3

COST

By using the development given in Chapter 2, a cost function which is a measure of the performance of an implementation of the topological structure may be defined. The objective of this chapter is to define such a cost function. Equations which express this cost function in terms of major design decisions (e.g., whether to use the display processor or a computer program as the picture generator, whether to use the structure $\{B,E/D,F'\}$ or $\{B,E/D,\mu,\gamma\}$, etc.) are derived in Chapter 4. These equations may be applied to determine the optimum implementation of the topological structure for a given hardware configuration and a given application.

The cost of an implementation of the structure will be considered to be the average time which is required to modify and interpret the structure in response to a control language input. For simplicity, the computer which controls the display processor will be assumed to be dedicated to the use of that display processor. Consequently, this response time is not affected by the activities of other users. The discussion will be restricted to display processors whose picture must be continually refreshed. Furthermore, in order to avoid momentary losses of the picture, the process of refreshing the picture will be assumed to have priority over the process of responding to control language inputs. Display processors whose

picture need not be continually refreshed will not be considered because the response time for these devices is independent of the picture generation process and is generally insignificant when compared to the time which is required for this process.

Certain constraints are imposed on the set of possible implementations by the hardware configuration. One of these is a limitation on the amount of storage which is available to represent the structure. Since the computer is assumed to be dedicated to the use of the display processor, this storage is the total storage in the computer minus that which is occupied by the program. A second constraint is a limitation on the amount of time which is available to produce a picture from the structure. This time is the smallest time interval which produces objectionable flicker (typically about 1/30 sec.).

In order to describe these constraints, as well as the cost of an implementation of the structure, both the storage which is required by the structure and the time which is arrested from the computer whenever the picture is generated must be described. These items are described in Sections 3.1 and 3.2 below. This discussion is followed in Section 3.3 by a discussion of the cost.

3.1 Storage Requirement

As has been discussed in Chapter 2, the topological structure may be represented by either of the sets $\{B,E/D,F'\}$ or $\{B,E/D,\mu,\gamma\}$.

Since the domain of each of the functions F', μ, and γ is B, each of these functions may be represented by storing one of its values as part of the representation of each element of B. If this is done, the storage which is required to represent either of the sets $\{B, E/D, F'\}$ or $\{B, E/D, \mu, \gamma\}$ is that necessary to represent the elements of B and E/D. In addition to the information which is needed to describe the structure, the set of subpictures which are associated with the elements of E/D must be stored so that a picture may be produced. Each of these subpictures may be stored as part of the representation of the class in E/D with which it is associated. For $e \in E_0$, $\beta(e)$ must be represented as plotting information, whereas, for $e \in E - E_0$, $\beta(e)$ may be represented by a reference to the set $\rho(\{e\} \times S(e)) \subset B$. Because subpictures are represented in these two different ways, the storage requirement for classes in E_0/D_0 will be expressed separately from that for classes in $E/D - E_0/D_0$. The total storage s which is required to represent the topological structure at any given time is then

$$s = |B| \bar{s}_b + |E_0/D_0| \bar{s}_{d_0} + |E/D - E_0/D_0| s_d, \qquad (3.1)$$

where

$\bar{s}_b$ is the average storage which is required to represent an element of B,

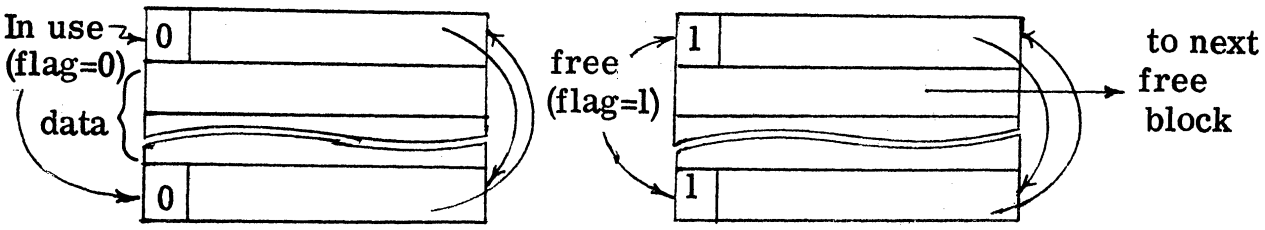
$\bar{s}_{d_0}$ is the average storage which is required to represent a class in E_0/D_0, and

s_d is the storage which is required to represent a class in $E/D - E_0/D_0$.

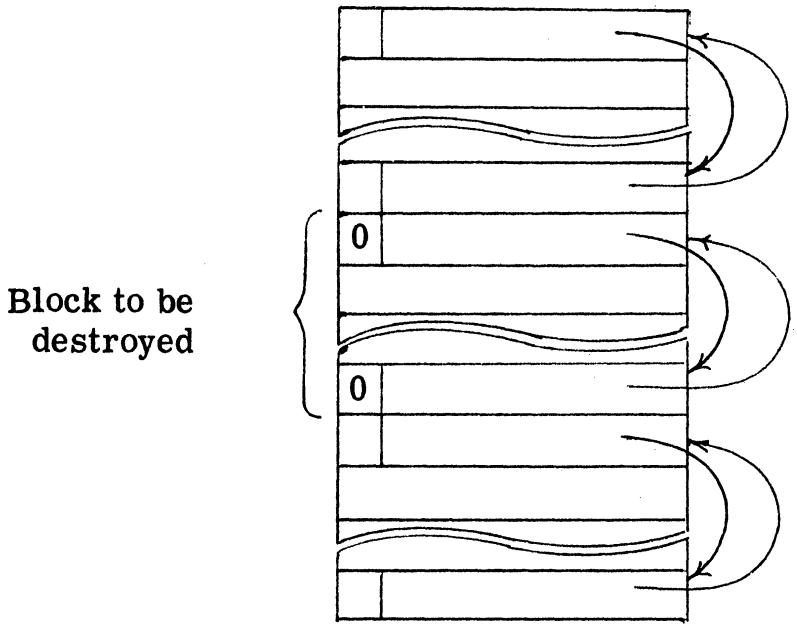
The average storage per element of B and the average storage per element of E_0/D_0 appear in this equation because the various elements of B generally require different amounts of storage and the various elements of E_0/D_0 generally require different amounts of storage. However, s_d does not represent an average because each element of $E/D - E_0/D_0$ generally requires the same amount of storage. The storage which is required to represent the elements of each of these sets is discussed below, following a discussion of the overhead storage which is common to all elements.

3.1.1 Overhead Storage

Since the purpose of the topological structure is to facilitate modification of the picture, the sets B and E/D are modified frequently during the execution of the program. Consequently, blocks of storage which are required to represent elements of these sets must be created (i.e., obtained from a free storage pool) and destroyed (i.e., returned to the free storage pool) frequently. In order to manage storage in this way, some overhead storage is generally associated with each block. For example, a simplified form of a dynamic storage allocation technique which has been described by Knuth [30 pp.435-442] is depicted in Figure 3.1. In addition to those locations which are needed to represent data, each block which is created by



a. Storage Blocks



b. A Storage Block To Be Destroyed

Figure 3.1

A Storage Allocation Scheme

this method must have a control word at each end of the block (Figure 3.1a). Each of these control words contains a pointer to the other end of the block and a flag to indicate whether or not the block is being used. Furthermore, each free block is linked to another free block by a pointer which occupies the first location which is used for data when this block is created. These pointers are arranged so that the free blocks may be examined sequentially.

In order to create a block of length n, the string of free storage blocks is scanned to find a block of length $m \geq n$. If $m \geq n$, but $m \leq n+2$, the block is removed from the free storage string and created. If $m > n+2$, the block is subdivided into two blocks, one of length n and one of length m-n. The block of length n is then created, and the block of length m-n is substituted for the original block of length m in the free storage string. (A block of length $m \leq n+2$ is not subdivided because a free storage block of length m-n cannot be formed.) The pointer in the first control word of each free block which is examined is used to determine the length of the block for this operation. This length is one more than the value of this pointer minus the address of the location in which it appears.

Full use of the control words is made when a block is destroyed (Figure 3.1 b). In order to insure that the free storage blocks are as large as possible, each block must be concatenated to any free blocks which are adjacent to it when it is destroyed. Whether or

not either of the two blocks which are adjacent to the block to be destroyed is free is determined by examining the flag in the last word of the block which precedes the block to be destroyed and the flag in the first word of the block which follows the block to be destroyed. If one of these flags is a 1, the pointer in the word in which it appears is used to locate the other control word in the same block. This control word, together with a control word in the block which is being destroyed, is then modified to concatenate the two blocks.

The overhead storage which is required to hold the control information for creating and destroying storage blocks is generally constant per block. If the amount of storage which is required by any element of $B \cup E/D$ is assumed not to vary between the time that that element is created and the time that it is destroyed, each of these elements requires only one block of storage. Consequently, this overhead storage is generally constant for all elements of $B \cup E/D$.

Additional overhead storage may be needed in each block to hold information which indicates whether that block represents an element of B, an element of $E/D - E_0/D_0$, or an element of E_0/D_0. The topological structure can be implemented in either of two ways: as a list structure, or as a display processor program. Whenever a list structure is used, the program which generates the picture, as well as programs which interpret the structure in order to respond to control language inputs, must determine the type of element that each block represents in order to interpret it properly.

However, if the topological structure is implemented as a display processor program, the display processor merely executes this program and does not need to interpret the meaning of each storage block. Although programs which operate on this structure do need this information, the format of each storage block usually may be examined to determine which type of element the block represents. For example, the program which is described in Appendix A determines which type of element a particular block represents by examining its first word. If this word contains a push jump command, the block represents an element of B, whereas, if it contains a jump command, the block represents an element of $E/D - E_0/D_0$. If this word contains neither a push jump command or a jump command, the block represents an element of E_0/D_0.

3.1.2 Storage Per Element of B

In order to describe the storage which is required to represent an element of B, the information which might be stored to represent an element of B is considered. Each element $\rho(e,e') \in B$ was defined to be a triple of the form $(D[e], D[e'], \eta(e,e'))$. The purpose of the function η was to distinguish distinct elements $(D[e], D[e'], \eta(e,e'))$ and $(D[e], D[e''], \eta(e,e''))$ for which $D[e'] = D[e'']$. If each element of B is stored in a random-access memory, the address of each element of B is unique and distinguishes it from other elements of B. Consequently, the function η need not be stored. Since each element of

E/D is generally a component of more than one element of B, the elements of E/D are stored independently of the elements of B. Then, references to the classes D[e] and D[e'] either must be stored as part of the representation of $\rho(e,e')$ or must be computable from other information in the structure.

As has been mentioned at the beginning of Section 3.1, a value of either the function F' or the function γ must be stored as part of the representation of each element of B to permit computation of coordinate transformations when the picture is generated. Which of these values is stored is determined by the design decision u_3, which will be discussed in Chapter 4. For the structure $\{B,E/D,F'\}$, the matrix $F'(\rho(e,e'))$ must be stored as part of the representation of $\rho(e,e')$. For the structure $\{B,E/D,\mu,\gamma\}$, the sequence $\gamma(\rho(e,e'))$ must be stored as part of the representation of $\rho(e,e')$. Although the function μ seems to be needed to describe this latter structure, in fact the values of μ need not be stored. Since each element $\mu(\rho(e,e')) \in B \cup E/D$ is encountered during the picture generation process before $\rho(e,e')$ is encountered, the information which the function μ contributes to the picture generation process may be retained until needed. In particular, since $\mu(\rho(e,e'))$ is examined before $\rho(e,e')$ is examined during the generation of the picture, the matrix $G''(\rho(e,\delta(e,e')))G(D[\delta(e,e')])F(e,\delta(e,e'))$ which is computed (as described in Section 2.2.2.3) when the entity $\delta(e,e')$ is displayed may be retained until $\rho(e,e')$ is examined. Then, this matrix may be premultiplied by $G'(\rho(e,e'))$ to produce $F(e,e')$.

However, a reference to $\mu(\rho(e,e'))$ may be included in the representation of $\rho(e,e')$ to facilitate interrogation of the structure for ordering information. Even if coordinate transformations are represented by values of F', this ordering information may be useful. For example, the ports of the transistor shown in Figure 2.4 may be ordered so that each of them can be identified as either the collector, base, or emitter of the transistor. The design decision u_8, to be discussed in Chapter 4, specifies whether or not references to values of μ are stored.

If values of μ are not stored in the blocks which represent elements of B, some other indication of ordering must be stored in these blocks so that these elements may be scanned to generate a picture. The function μ: B $\rightarrow$ B $\cup$ E/D has been defined in Chapter 2 such that $\mu(\rho(e,e'))$ denotes the element of $\rho(\{e\} \times S(e))$ immediately before $\rho(e,e')$ which is needed to form $\beta(e)$ (unless $\rho(e,e')$ is the first element of $\rho(\{e\} \times S(e))$ which is needed to form $\beta(e)$). However, in order to determine the element of $\rho(\{e\} \times S(e))$ immediately after $\rho(e,e')$ which is needed to form $\beta(e)$, the structure must be searched if the function μ is the only indication of ordering which is stored. For this reason, a reference to a value of the function μ': B $\rightarrow$ B $\cup$ E/D is generally stored in each block which represents an element of B, where

$$\mu'(\rho(e,e')) = \begin{cases} D[e], \text{ if } \delta'(e,e') = e', \text{ or} \\ \\ \rho(e, \delta'(e,e')), \text{ otherwise} \end{cases} \qquad (3.2)$$

Then, if $\mu'(\rho(e,e')) \in E/D$, $\rho(e,e')$ is the last element of $\rho(\{e\}\times S(e))$ which is needed to form $\beta(e)$, whereas, if $\mu'(\rho(e,e')) \in B$, $\mu'(\rho(e,e'))$ is the element of $\rho(\{e\} \times S(e))$ immediately after $\rho(e,e')$ which is needed to form $\beta(e)$.

In order to either respond to a control language input or to enforce a constraint, elements of the set

$$\mathbb{P}(D[e]) = \{\rho(e',e'') \mid e'' \in D[e] \text{ and } (e',e'') \in A\} \qquad (3.3)$$

must often be determined for a class $D[e]$. For example, if a class $D[e']$ is to be substituted for a class $D[e]$, the second component of each element of $P(D[e])$ must be modified. (Recall that $\rho(e',e'') = (D[e'], D[e''], \eta(e',e''))$.) As a second example, the elements of $P(D[e])$ must be destroyed in order to remove a connection e from the network diagram shown in Figure 2.4. In order to facilitate operations on the structure such as these, a reference to each element of $P(D[e])$ may be stored in the structure. However, because elements of B are frequently added to and removed from each set $P(D[e])$, these references are generally not stored as part of the representation of $D[e]$. Instead, the elements of each set $P(D[e])$ are generally linked together in a ring with the class $D[e]$. References to adjacent elements in this ring must then be stored as part of the representation of each element of $P(D[e])$. Whether or not these elements are linked together in a ring is determined by the design decision u_{10}, which will be discussed in Chapter 4.

Thus, the items which may be stored to represent $\rho(e,e') \in B$ are the following:

(1) A reference to the class D[e],

(2) A reference to the class D[e'],

(3) A reference to $\mu(\rho(e,e'))$,

(4) A reference to $\mu'(\rho(e,e'))$,

(5) Either the matrix $F'(\rho(e,e'))$ or the sequence $\gamma(\rho(e,e'))$, and

(6) References to adjacent elements in the ring which links the elements of P(D[e]) with D[e'].

However, not all of these items need be stored in order to represent an element of B, since some of them may be computed from those which are stored. The second example shown in Section 3.1.4 below illustrates the omission of some of these items.

In addition to the overhead storage and the storage which is needed to describe an element of B, some storage which contains no useful information may be needed to represent an element of B. If two or more of the items which describe an element of B each do not occupy an integer number of storage locations, they may share a common storage location. The remaining storage in that location may then be unusable because either it is not large enough to hold another item or there are no other items to be stored.

The average storage $\bar{s}_b$ is the sum of the overhead storage, the average storage which is required to describe an element of B, and this unusable storage. This sum may be expressed as follows:

$$\bar{s}_b = s_0 + s_t + s_p + s_e + s_e' + s_\mu + s_\mu' + \bar{s}_f + \bar{s}_\gamma + \bar{s}_{wb}, \tag{3.4}$$

where

s_0 is the overhead storage which is associated with creating and destroying a storage block,

s_t is the overhead storage which is needed to indicate that the storage block represents an element of B,

s_p is the storage which is required in each block which represents an element of B to reference adjacent blocks in the ring which represents a set P(D[e]),

s_e is the storage which is required to reference the first component of an element of B,

s'_e is the storage which is required to reference the second component of an element of B,

s_μ is the storage which is required to reference a value of μ,

s'_μ is the storage which is required to reference a value of μ'.

$\bar{s}_f$ is the average storage which is required to represent a value of F',

$\bar{s}_\gamma$ is the average storage which is required to represent a value of γ, and

$\bar{s}_{wb}$ is the average storage which is unusable in each block which represents an element of B.

In practice, several of the terms in this equation are zero. For example, if the structure is $\{B,E/D,F'\}$, $\bar{s}_{\gamma} = 0$, whereas, if the structure is $\{B,E/D,\mu,\gamma\}$, $\bar{s}_{f} = 0$. Furthermore, regardless of which structure is used, the terms s_p, s_e, and s_{μ} are zero if each set P(D[e]), first components of elements of B, and values of μ are determined by searching the structure.

As indicated by the fact that $\bar{s}_f$ represents an average amount of storage, various amounts of storage may be required to represent values of the function F'. In particular, various amounts of storage are required if default values are assumed for those elements of each matrix which are not specified. For example, consider a two-dimensional display system which uses coordinate transformations which are represented by matrices of the form

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & 0 & \sigma \end{bmatrix}.$$

This matrix represents a coordinate transformation which maps coordinates expressed with respect to a coordinate system W into coordinates expresses with respect to a coordinate system W' whose origin is the point $(-\Delta x, -\Delta y)$ in W, whose x and y axes form the angle θ with the respective axes of W, and whose units of measurement are σ times as large as those of W. The matrix may be stored as the

four-tuple $(\Delta x, \Delta y, \theta, \sigma)$. If the default value of each matrix element is assumed to be the corresponding element of the identity matrix, not all components of this four-tuple need be stored. If $\sigma = 1$, the three-tuple $(\Delta x, \Delta y, \theta)$ is sufficient to represent the matrix, and if both $\sigma = 1$ and $\theta = 0$, the pair $(\Delta x, \Delta y)$ is sufficient. If $\sigma = 1$, $\theta = 0$, and $\Delta x = \Delta y = 0$, the matrix is the identity matrix and need not be stored at all.

Similarly, various amounts of storage may be required to represent values of the function γ. Each sequence $\gamma(\rho(e,e'))$ assumes one of four forms: $(I, G(D[e']), I)$, $(I, G(D[e']), G(D[e'])^{-1})$, $(G(D[e'])^{-1}, G(D[e']), I)$, or $(G(D[e'])^{-1}, G(D[e']), G(D[e'])^{-1})$. The matrix $G(D[e'])$ is not stored, for the coordinate transformation which it represents is performed when the subpicture $\beta(e')$ is drawn. Because there may be many elements of B whose second component is $D[e'], G(D[e'])^{-1}$ is not stored with $\gamma(\rho(e,e'))$. The only information which is needed to represent the sequence $\gamma(\rho(e,e'))$ is then a specification of its form. In some implementations (e.g., the program described in Appendix A), the amount of storage which this specification requires depends on which form $\gamma(\rho(e,e'))$ assumes.

The dependence of the terms in this equation on the design decisions will be discussed in Chapter 4. In that chapter, a method for computing $\bar{s}_b$ without actually determining $\bar{s}_{wb}$ will also be described.

3.1.3 Storage Per Element of E/D

In order to describe the storage which is required to represent a class in E/D, the information which is necessary to represent this class is considered. Since each entity is to be computed from the structure, the entities in each class in E/D need not be stored. However, the subpicture $\beta(e)$ must be included in the representation of $D[e]$ so that the entities in $D[e]$ may be generated. For $D[e] \in E/D - E_0/D_0$, this subpicture may be represented by a reference to the set $\rho(\{e\} \times S(e))$. However, a reference to $\mu'(\rho(e,e'))$ is assumed to be stored in the block which represents each element $\rho(e,e')$ so that the element of $\rho(\{e\} \times S(e))$ immediately after $\rho(e,e')$ which is needed to produce $\beta(e)$ can be easily identified. Consequently, a reference to $\rho(e,e')$ such that $\mu(\rho(e,e')) = D[e]$ (i.e., the first element of $\rho(\{e\} \times S(e))$ which is needed to form $\beta(e)$) is sufficient to represent this set. For $D[e] \in E_0/D_0$, however, the subpicture $\beta(e)$ must be represented explicitly.

In addition to the subpicture $\beta(e)$, other information may be stored to represent each class $D[e]$. As has been mentioned above in Section 3.1.1, the elements of the set $P(D[e])$ may be linked together in a ring with $D[e]$. If $P(D[e])$ is represented in this way, some storage is required in the block which represents $D[e]$ to reference blocks which are adjacent to it in the ring. The amount of storage which is required for this purpose is s_p, the same as that

which is required for this purpose in each block which represents an element of B.

The matrix $G(D[e])^{-1}$ must also be stored as part of the representation of some of the classes $D[e]$ if the structure is $\{B, E/D, \mu, \gamma\}$. If a subpicture $\beta(e')$ for which $(e, e') \in C$ is modified so that $G(D[e'])$ is modified, $G(D[e])$ must also be modified. Consequently, if each matrix $G(D[e])^{-1}$ is stored with $D[e]$ for $D[e] \in E/D - E_0/D_0$, the computer must modify this matrix whenever it modifies a subpicture on which the matrix depends. This requirement produces a demand on computer time which defeats the purpose of the structure $\{B, E/D, \mu, \gamma\}$. However, if each matrix $G(D[e])^{-1}$ is stored explicitly for $D[e] \in E_0/D_0$, but is computed from the structure when the picture is generated for $D[e] \in E/D - E_0/D_0$, it is effectively modified whenever a subpicture on which it depends is modified.

In order to describe the computation of each matrix $G(D[e])^{-1}$ for $D[e] \in E/D - E_0/D_0$, recall from Equation 2.27 that

$$G(D[e]) = G''(\rho(e, e_n))G(D[e_n])G'(\rho(e, e_n)) \ldots$$
$$G''(\rho(e, e_2))G(D[e_2])G'(\rho(e, e_2))G''(D(e, e_1))$$
$$G(D[e_1])G'(\rho(e, e_1)),$$

where $S(e) = \{e_1, e_2, \ldots, e_n\}$, $\delta(e, e_1) = e_1$, and $\delta(e, e_{i+1}) = e_i$ for $i = 1, 2, \ldots, n-1$. From this equation,

$$G(D[e])^{-1} = G'(\rho(e,e_1))^{-1}G(D[e_1])^{-1}G''(\rho(e,e_1))^{-1}G'(\rho(e,e_2))^{-1}$$
$$G(D[e_2])^{-1}G''(\rho(e,e_2))^{-1} \ldots$$
$$G'(\rho(e,e_n))^{-1}G(D[e_n])^{-1}G''(\rho(e,e_n))^{-1} . \qquad (3.5)$$

If $G(D[e'])^{-1}$ is stored explicitly for all $D[e'] \in E_0/D_0$, then $G(D[e])^{-1}$ for $D[e] \in E/D - E_0/D_0$ may be obtained from the structure by repeated application of this equation. If this is done, and a subpicture on which $G(D[e])^{-1}$ depends is modified, $G(D[e])^{-1}$ assumes the correct value with no further adjustment of the structure.

If the display processor is used as the picture generator (as determined by the design decision u_1, which is described in Chapter 4), some information which is needed only to control the display processor often must be stored in each block which represents an element E/D. In particular, display processor commands which cause the display processor to compute $G(D[e])^{-1}$ for $D[e] \in E/D - E_0/D_0$ must be included in the block which represents D[e] in the topological structure $\{B, E/D, \mu, \gamma\}$. The program which is described in Appendix A was designed to manage a topological structure in which a display processor jump command is used for this purpose. As a second example, the reference to D[e'] which is stored in the block which represents $\rho(e,e')$ may be implemented as a display processor subroutine call. Display processor commands which return from this subroutine call must then be stored in each block which represents a class in E/D.

The average storage $\bar{s}_{d0}$ which is required to represent an element of E_0/D_0 may be expressed as follows:

$$\bar{s}_{d0} = s_0 + s_t + s_p + s_{r0} + \bar{s}_{\beta 0} + \bar{s}_g + \bar{s}_{wd0}, \tag{3.6}$$

where

s_{r0} is the storage per element of E_0/D_0 which is occupied by information whose only purpose is to control the display processor,

$\bar{s}_{\beta 0}$ is the average storage per element of E_0/D_0 which is required to represent a subpicture as plotting information,

$\bar{s}_g$ is the average storage per element $D[e] \in E_0/D_0$ which is required to represent the matrix $G(D[e])^{-1}$, and

s_{wd0} is the average unusable storage per element of E_0/D_0.

Similarly, the storage s_d which is required to represent an element of $E/D - E_0/D_0$ may be expressed as follows:

$$s_d = s_0 + s_t + s_p + s_r + s_\beta + s_{wd}, \tag{3.7}$$

where

s_r is the storage per element of $E/D - E_0/D_0$ which is occupied by information whose only purpose is to control the display processor,

s_β is the storage per element $D[e] \in E/D - E_0/D_0$ which is occupied by references to the set $\rho(\{e\} \times S(e))$, and

s_{wd} is the unusable storage per element of $E/D - E_0/D_0$.

Like the terms in Equation 3.4, some of the terms in Equations 3.6 and 3.7 may be zero for some implementations. For the structure

$\{B, E/D, F'\}$, $\bar{s}_g = 0$. Furthermore, $s_p = 0$ if each set $P(D[e])$ is determined by searching the structure. If the display processor is not used as the picture generator, $s_{r0} = s_r = 0$.

The terms in Equations 3.6 and 3.7 will be described as functions of the design decisions in Chapter 4. In that chapter, a method for computing $\bar{s}_{d0}$ and s_d without actually determining $\bar{s}_{wd0}$ or s_{wd} will also be described.

3.1.4 Examples

In order to illustrate the storage requirements for elements of B and elements of E/D, a list structure representation of $\{B, E/D, \mu, \gamma\}$ is depicted in Figures 3.2 and 3.3. Figure 3.2 shows the form of the data blocks which represent elements of B and E/D, whereas Figure 3.3 shows the interconnection of these blocks to represent a picture which consists of two angles (e_1 and e_2) which share a common side (e_3). In this example, each storage location is considered to be capable of containing two pointers. Consequently, $s_p = s_e = s'_e = s_\mu = s'_\mu = 0.5$ locations. Because this list structure represents $\{B, E/D, \mu, \gamma\}$, $\bar{s}_f = 0$. Each sequence $\gamma(\rho(e, e'))$ is represented by two bits as follows:

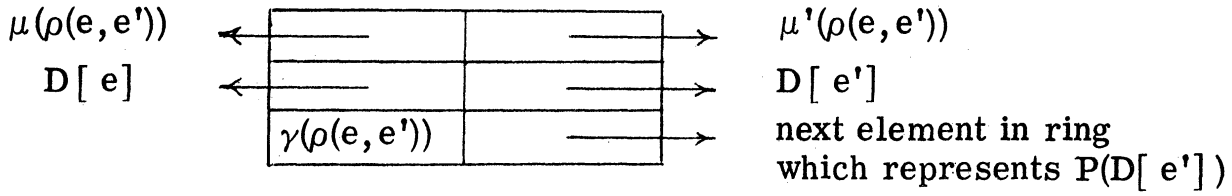
00: $\gamma(\rho(e,e')) = (I, G(D[e']), I)$

01: $\gamma(\rho(e,e')) = (I, G(D[e']), G(D[e'])^{-1})$

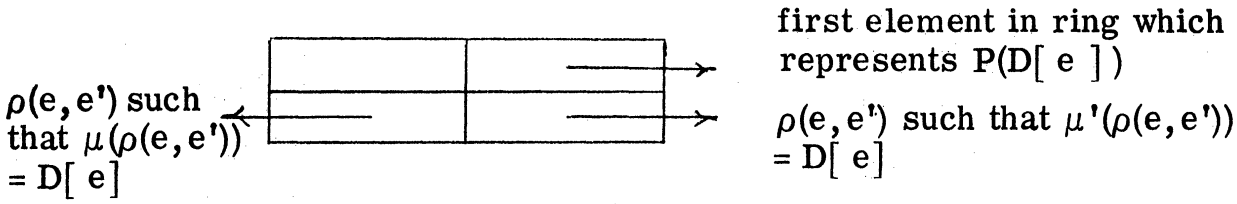
10: $\gamma(\rho(e,e')) = (G(D[e'])^{-1}, G(D[e']), I)$

11: $\gamma(\rho(e,e')) = (G(D[e'])^{-1}, G(D[e']), G(D[e'])^{-1})$

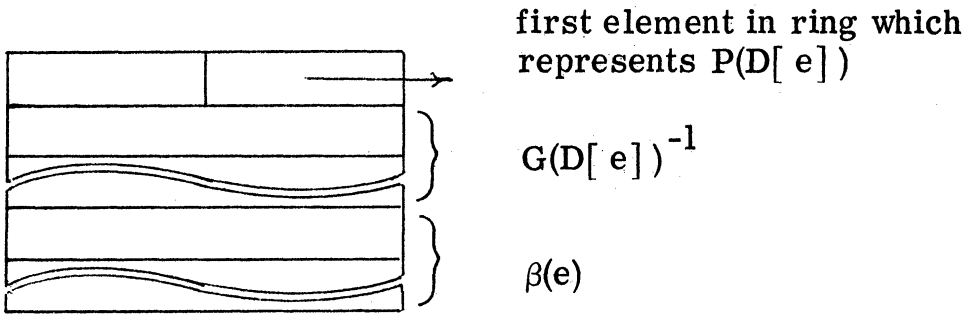
Consequently, $\bar{s}_\gamma = 2$ bits. Since no other item which is stored to represent an element of B can be packed with these two bits, the



a. A Block Which Represents $\rho(e,e')$



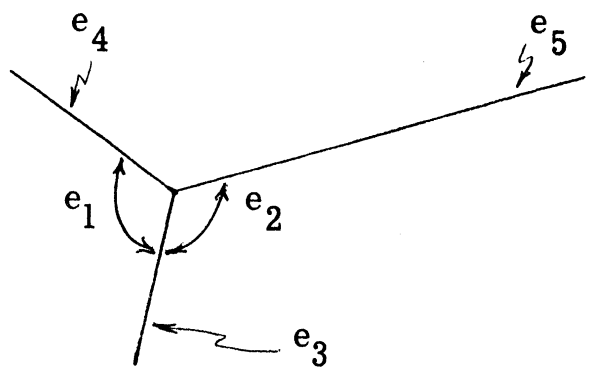
b. A Block Which Represents $D[\ e] \in E/D\text{-}E_0/D_0$



c. A Block Which Represents $D[\ e] \in E_0/D_0$

Figure 3.2

Data Blocks for a List Structure Representation of $\{B, E/D, \mu, \gamma\}$



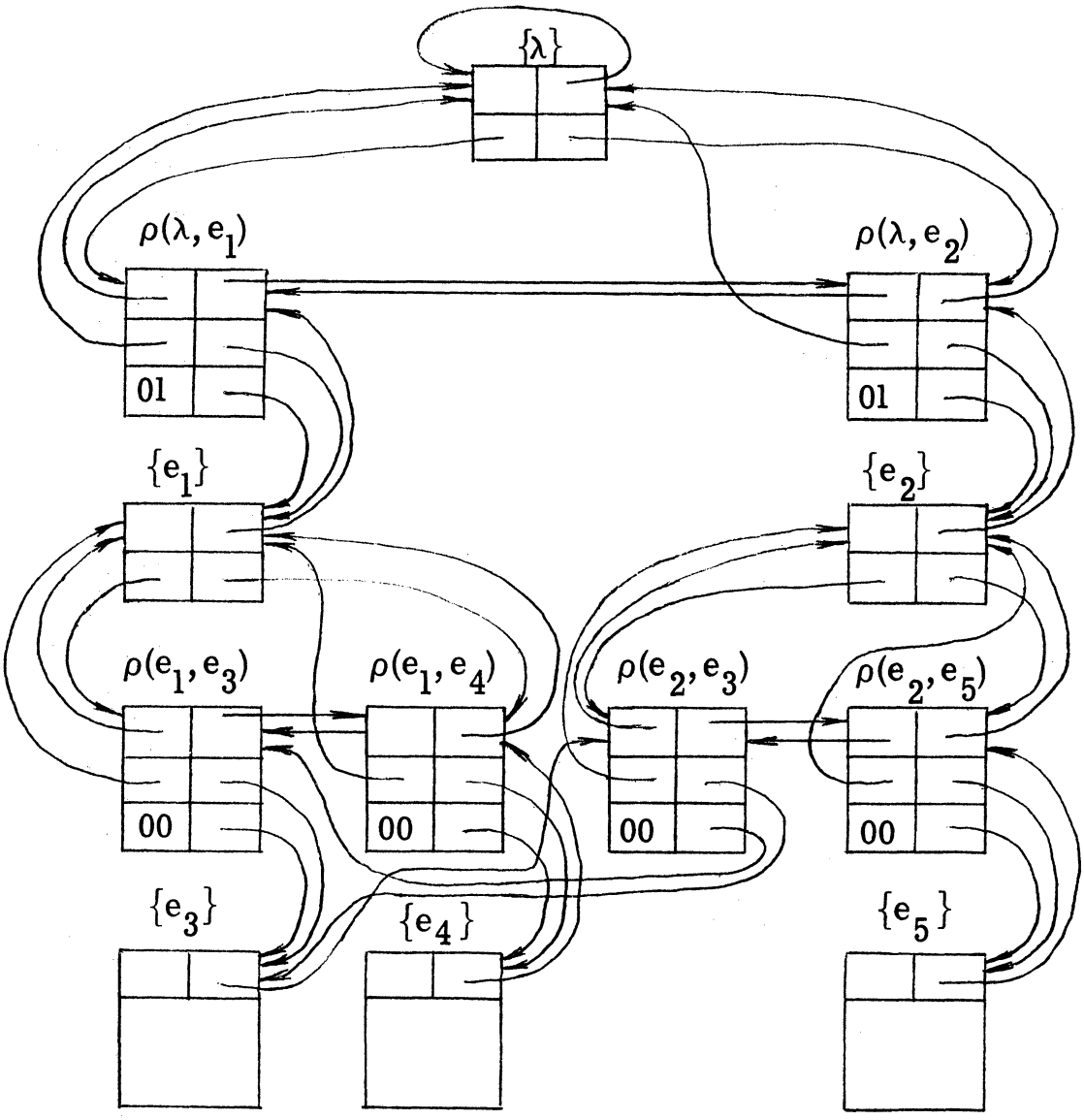


Figure 3.3

Representation of Two Angles by a List Structure

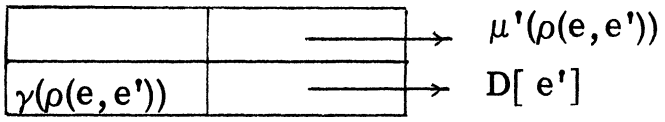
remainder of the location in which each value of γ is stored is unusable, and $\bar{s}_{wb} = 0.5$ locations - 2 bits. For $D[e] \in E/D - E_0/D_0$, $\beta(e)$ is represented by pointers to both the first and last elements of $\rho(\{e\} \times S(e))$. Consequently, $s_\beta = 1$ location. For $D[e] \in E_0/D_0$, $\beta(e)$ and $G(D[e])^{-1}$ are stored explicitly. Since there is no information to store in the high order half of the first word in each block which represents a class $D[e]$, $\bar{s}_{wd0} = s_{wd} = 0.5$ locations. For any class $D[e] \in E_0/D_0$, the storage which is required to represent $G(D[e])^{-1}$ plus that which is required to represent $\beta(e)$ is assumed to be at least 3 locations. Whether a block represents an element of B, $E/D - E_0/D_0$, or E_0/D_0 may then be easily determined from the length of the storage block. Each block of length 3 represents an element of B, each block or length 2 represents an element of $E/D - E_0/D_0$, and each block of length ≥ 4 represents an element of E_0/D_0. Because the type of element which a block represents may be determined from its length, $s_t = 0$.

Much of the information which is stored in this list structure may be computed from other information in the structure. Consequently, by eliminating these items from the structure, computation time may be traded for storage. The pointers which represent values of μ may be eliminated at the expense of the computation time which is required to sequence through portions of the structure whenever a value of this function is to be determined. Similarly, the pointer to $\rho(e,e')$

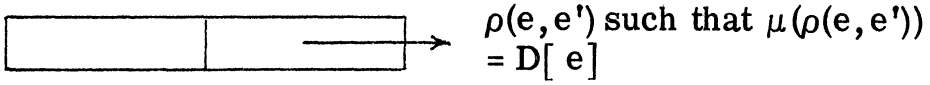
such that $\mu'(\rho(e,e')) = D[e]$ may be eliminated from each block which represents a class $D[e]$. The pointer to the block which represents $D[e]$ in each block which represents an element $\rho(e,e') \in B$ may also be determined by searching a portion of the structure. Consequently, this pointer may also be eliminated. The pointers which link the elements of each set $P(D[e])$ with $D[e]$ may be eliminated at the expense of the time which is required to search the entire structure for this information.

If all of these simplifications are applied to the list structure shown in Figure 3.2, the list structure shown in Figure 3.4 results. For this structure, $s'_e, s'_\mu, \bar{s}_\gamma, \bar{s}_g$, and s_{wd} are unchanged from their values for the list structure described above. However, $s_p = s_e = s_\mu = 0$ and $s_\beta = 0.5$ locations. Furthermore, $\bar{s}_{wb} = 1$ location - 2 bits, and $\bar{s}_{wd0} = 0$.

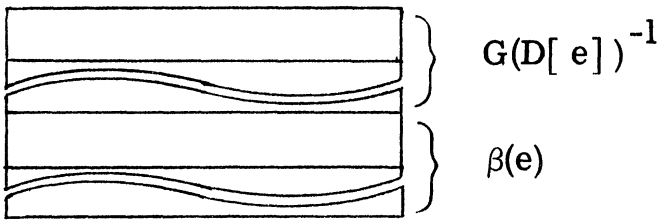
Both of the list structures shown in Figures 3.2 and 3.4 represent the topological structure $\{B, E/D, \mu, \gamma\}$. For many applications, the latter list structure cannot be interpreted as rapidly as the former list structure. However, the former list structure may require more time to modify because more information must be changed when it is modified. In order to choose between alternatives such as these, the cost will be expressed in Chapter 4 as a function of the information which is stored.



a. A Block Which Represents $\rho(e,e')$



b. A Block Which Represents $D[\ e] \in E/D\text{-}E_0/D_0$



c. A Block Which Represents $D[\ e] \in E_0/D_0$

Figure 3.4

Modification of the List Structure Blocks in Figure 3.2

3.2 Computer Time Required to Generate a Picture

As has been mentioned in Chapter 2, the picture λ is generated by displaying all entities in E_0. However, in order to determine entities in E_0 from the structure, the picture generator must sequence through the portion of the structure which describes entities in $E-E_0$. As has been mentioned in Chapter 1, either the computer or the display processor may perform this function. (Which method is used is determined by the design decision u_1, which is described in Chapter 4.) If the computer sequences through the structure, it must be devoted to this process for short periods of time during which it cannot perform other functions (such as responding to control language inputs). Even if the display processor sequences through the structure, the computer may be suspended while the display processor accesses its memory. Similarly, once an entity in E_0 has been determined from the structure, the computer must be suspended while the entity is transferred to the display processor. In this section, the time during which the computer must be devoted to the process of generating the picture is described.

3.2.1 An Algorithm for Generating the Picture

In order to describe the time during which the computer must be devoted to the generation of the picture, the procedure which is used must be considered. A procedure which is applicable to either of the structures $\{B,E/D,F'\}$ or $\{B,E/D,\mu,\gamma\}$ is given by Algorithm 3.1 below. Although other algorithms are possible, the algorithms which have been used in practice differ insignificantly from this one.

Consequently, in all subsequent discussion, Algorithm 3.1 will be assumed to describe the procedure which is used to generate the picture.

Algorithm 3.1

Procedure for generating a picture from either of the topological structures $\{B, E/D, F'\}$ or $\{B, E/D, \mu, \gamma\}$.

(1) Let $J = H(\lambda) = I$, let b be the element of B such that $\mu(b) = \{\lambda\}$, and empty a push-down stack for use in this algorithm.

(2) Save b on the push-down stack. Let $X \in E/D$ be the second component of b. If the topological structure is $\{B, E/D, F'\}$, let $J = F'(b)J$; otherwise, let $J = G'(b)J$. If there exists an element $b' \in B$ such that $\mu(b') = X$, let $b = b'$ and repeat this step.

(3) J now equals H(e) for some entity in E_0. Display this entity by applying J to the subpicture which is associated with X. If the topological structure is $\{B, E/D, \mu, \gamma\}$, let $J = G(X)J$ after this entity is displayed.

(4) If the push-down stack is empty, terminate. Otherwise, restore b from the push-down stack. If the topological structure is $\{B, E/D, F'\}$, let $J = F'(b)^{-1}J$; otherwise, let $J = G''(b)J$. If $\mu'(b) \in B$, let $b = \mu'(b)$ and proceed with Step 2. Otherwise, repeat this step. ∎

In order to illustrate this algorithm, the steps which are needed to generate the picture which is shown in Figure 2.1, whose relation A is shown in Figure 2.3, and whose set B is shown in Figure 2.10 are given below. Coordinate transformations are represented by values of F', and the following values of μ are assumed:

$$\mu(\rho(\lambda, e_5)) = D[\lambda] = \{\lambda\},$$

$$\mu(\rho(\lambda, e_4)) = \rho(\lambda, e_5),$$

$$\mu(\rho(\lambda, e_6)) = \rho(\lambda, e_4),$$

$$\mu(\rho(e_4, e_1)) = D[e_4] = \{e_4, e_6\},$$

$$\mu(\rho(e_4, e_2)) = \rho(e_4, e_1), \text{ and}$$

$$\mu(\rho(e_4, e_3)) = \rho(e_4, e_2) .$$

Each step is represented below by its number and the items which are affected by performing it.

(1) $J = I$

$b = \rho(\lambda, e_5)$

(2) Stack contains $\rho(\lambda, e_5)$

$X = \{e_5\}$

$J = F'(\rho(\lambda, e_5)) = H(e_5)$

(3) Display e_5

(4) $b = \rho(\lambda, e_5)$; Stack empty

$J = I$

$b = \rho(\lambda, e_4)$

(2) Stack contains $\rho(\lambda, e_4)$

$X = \{e_4, e_6\}$

$J = F'(\rho(\lambda, e_4))$

$b = \rho(e_4, e_1)$

(2) Stack contains $\rho(\lambda, e_4), \rho(e_4, e_1)$

$X = \{e_1, e_2, e_7, e_8\}$

$J = F'(\rho(e_4, e_1))F'(\rho(\lambda, e_4)) = H(e_1)$

(3) Display e_1

(4) $b = \rho(e_4, e_1)$; Stack contains $P(\lambda, e_4)$

$J = F'(\rho(\lambda, e_4))$

$b = \rho(e_4, e_2)$

(2) Stack contains $\rho(\lambda, e_4), \rho(e_4, e_2)$

$X = \{e_1, e_2, e_7, e_8\}$

$J = F'(\rho(e_4, e_2))F'(\rho(\lambda, e_4)) = H(e_2)$

(3) Display e_2

(4) $b = \rho(e_4, e_2)$; Stack contains $\rho(\lambda, e_4)$

$J = F'(\rho(\lambda, e_4))$

$b = \rho(e_4, e_3)$

(2) Stack contains $\rho(\lambda, e_4), \rho(e_4, e_3)$

$X = \{e_3, e_4\}$

$J = F'(\rho(e_4, e_3))F'(\rho(\lambda, e_4)) = H(e_3)$

(3) Display e_3

(4) $b = \rho(e_4, e_3)$; Stack contains $\rho(\lambda, e_4)$

$J = F'(\rho(\lambda, e_4))$

(4) $b = \rho(\lambda, e_4)$; Stack empty

$J = I$

$b = \rho(\lambda, e_6)$

(2) Stack contains $\rho(\lambda, e_6)$

$X = \{e_4, e_6\}$

$J = F'(\rho(\lambda, e_6))$

$b = \rho(e_6, e_7)$

(2) Stack contains $\rho(\lambda, e_6), \rho(e_6, e_7)$

$X = \{e_1, e_2, e_7, e_8\}$

$J = F'(\rho(e_6, e_7))F'(\rho(\lambda, e_6)) = H(e_7)$

(3) Display e_7

(4) $b = \rho(e_6, e_7)$; Stack contains $\rho(\lambda, e_6)$

$J = F'(\rho(\lambda, e_6))$

$b = \rho(e_6, e_8)$

(2) Stack contains $\rho(\lambda, e_6), \rho(e_6, e_8)$

$X = \{e_1, e_2, e_7, e_8\}$

$J = F'(\rho(e_6, e_8))F'(\rho(\lambda, e_6)) = H(e_8)$

(3) Display e_8

(4) $b = \rho(e_6, e_8)$; Stack contains $\rho(\lambda, e_6)$

$J = F'(\rho(\lambda, e_6))$

$b = \rho(e_6, e_3)$

(2) Stack contains $\rho(\lambda, e_6), \rho(e_6, e_9)$

$X = \{e_3, e_9\}$

$J = F'(\rho(e_6, e_9))F'(\rho(\lambda, e_6)) = H(e_9)$

(3) Display e_9

(4) $b = \rho(e_6, e_9)$; Stack contains $\rho(\lambda, e_6)$

$J = F'(\rho(\lambda, e_6))$

(4) $b = \rho(\lambda, e_6)$; Stack empty

$J = I$

Although each modification of the matrix J has been described as a matrix multiplication in this algorithm, some of these modifications may be performed in other ways. In particular, if the structure is $\{B, E/D, F'\}$, letting $J = F'(b)^{-1}J$ in Step 4 merely restores J to the value which it assumed before it was premultiplied by F'(b) in Step 2. In practice, the original value of J is often saved with b on the push-down stack during Step 2. This value is then retrieved later during Step 4 in lieu of premultiplying J by $F'(b)^{-1}$. Whether or not these matrices are saved on the push-down stack and later restored from the push-down stack is determined by the design decision u_5, which is discussed in Chapter 4.

For the structure $\{B, E/D, \mu, \gamma\}$, letting $J = G(X)J$ in Step 3 of the algorithm may also be accomplished without performing a matrix multiplication. Instead, the process of displaying an entity by applying J to the subpicture which is associated with X is used to modify J. For example, consider a two-dimensional display processor which contains x and y coordinate registers, an angle register, and a scale register. The contents of each of these registers will be denoted x, y, θ

and σ, respectively. The values x and y define a position relative to the coordinate system with respect to which the picture is to be generated, whereas θ and σ describe how the x and y registers are to be incremented when drawing vectors. The following display processor commands affect these registers:

(1) Rotate by $\Delta\theta$: $\Delta\theta$ is added to the content of the angle register, and the result if left in the angle register.

(2) Scale by $\Delta\sigma$: $\Delta\sigma$ is multiplied by the content of the scale register, and the result is left in the scale register.

(3) Draw vector with components Δx and Δy : $(\Delta x\cos\theta - \Delta y\sin\theta)/\sigma$ is added to the content of the x register and the result is left in the x register, $(\Delta x \cos\theta + \Delta y \sin\theta)/\sigma$ is added to the content of the y register and the result is left in the y register. A vector is drawn from the point whose coordinates were the original contents of the x and y registers to the point whose coordinates are the final contents of these registers.

The values x, y, θ, and σ are considered to represent the coordinate transformation matrix

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma x & \sigma y & 0 & \sigma \end{bmatrix}$$

In order to show how J is effectively premultiplied by G(X) by drawing the subpicture which is associated with X, the effect of each command on this matrix is considered. The rotate command increments the value θ, which is equivalent to performing the matrix multiplication

$$\begin{bmatrix} \cos\Delta\theta & \sin\Delta\theta & 0 & 0 \\ -\sin\Delta\theta & \cos\Delta\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma x & \sigma y & 0 & \sigma \end{bmatrix} = \begin{bmatrix} \cos(\theta+\Delta\theta) & \sin(\theta+\Delta\theta) & 0 & 0 \\ -\sin(\theta+\Delta\theta) & \cos(\theta+\Delta\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma x & \sigma y & 0 & \sigma \end{bmatrix}$$

The scale command multiplies the value σ, which is equivalent to the matrix multiplication

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \Delta\sigma \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma x & \sigma y & 0 & \sigma \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma\Delta\sigma x & \sigma\Delta\sigma y & 0 & \sigma\Delta\sigma \end{bmatrix}$$

Finally, the draw vector command increments the x and y registers as described, which is equivalent to the matrix multiplication

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta x & \Delta y & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & \sigma \\ -\sin\theta & \cos\theta & 0 & \sigma \\ 0 & 0 & 1 & 0 \\ \sigma x & \sigma y & 0 & \sigma \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sigma(x+(\Delta x\cos\theta - \Delta y\sin\theta)/\sigma) & \sigma(y+(\Delta x\cos\theta+\Delta y\sin\theta)/\sigma) & 0 & \sigma \end{bmatrix}$$

Consequently, the rotate, scale, and draw vector commands effectively premultiply the transformation matrix which is represented in the registers by matrices which represent rotation, scale change, and translation, respectively. In order to perform Step 3 of Algorithm 3.1, the x,y,angle,and scale registers are initially loaded with values which represent J. The contents of these registers are then modified by the rotate, scale, and draw vector commands which draw the subpicture which is associated with X. The final values which are contained in these registers then represent the transformation matrix G(X)J. Other matrix multiplications in this algorithm may also be performed in a similar way by the display processor for the structure $\{B,E/D,F'\}$. However, if the structure is $\{B,E/D,\mu,\gamma\}$, Algorithm 3.2, which is described later in Section 3.2.2.2, must be executed in order to perform one of these multiplications.

3.2.2 Time Required to Perform Algorithm 3.1

The time g which is required to perform Algorithm 3.1 is the total time during which the computer is devoted to the process of generating the picture. Note that g does not represent the total time which elapses during this process and does not depend on the plotting rate of the display processor. However, the plotting rate does affect the response time and will be considered later in Section 3.3. An expression for g is derived below by considering each step of Algorithm 3.1 in detail.

3.2.2.1 Initialization

The first step of the algorithm is performed only once during

the generation of the picture, since it represents an initialization. Because J is merely initialized, rather than obtained as the result of a matrix multiplication, the time which is required for this step is not large when compared to the time which is required to perform either Step 2 or Step 4. Furthermore, the time which is required for this step is not large when compared to the total time during which the computer is suspended while an entity in E_0 is transferred to the display processor in Step 3. Since the initialization step is performed only once, whereas other steps are generally performed many times, and since the time which it requires is not large relative to the time which is required to perform each other step, the time which is required for initialization is generally insignificant and may be ignored.

3.2.2.2 Identifying Entities From the Structure

The purpose of the second step of Algorithm 3.1 is to determine the information which is necessary to display an entity $e \neq \lambda$. As was mentioned in Chapter 2, the required information is the matrix H(e) and the subpicture $\beta(e)$. This step of the algorithm sets the matrix J to the value H(e) and sets X to D[e]. Since the only entities which are to be displayed are those in E_0, and since the subpicture $\beta(e)$ is represented explicitly as part of the representation of D[e] for each $e \in E_0$, D[e] is sufficient to describe $\beta(e)$ for each e which is displayed. For each entity $e \in E - E_0 - \{\lambda\}$, H(e) is computed as a step toward computing H(e') for some entities $e' \in D_0$, and D[e] is identified so that the next element of B which is to be examined can be identified.

Although all required information about an entity $e \neq \lambda$ is obtained each time that this step is performed, the number of times that the step is performed may be greater than $|\mathbb{E}-\{\lambda\}|$. In particular, if there exist distinct entities e, e', and e'' such that $e \in S(e') \cap S(e'')$, this step is performed to determine H(e) and D[e] at least twice during the generation of the picture: once when e is encountered as a part of e', and once when e is encountered as part of e''. Furthermore, H(e''') and D[e'''] are determined at least twice for each entity e''' such that there exists a path from e to e'''.

In order to describe the number of times that Step 2 of the algorithm is performed, the function ψ: E$\longrightarrow \{1, 2, \ldots\}$ is defined such that $\psi(e)$ is the number of times that e is displayed by Algorithm 3.1, either explicitly (if $e \in E_0$) or as a collection of entities in E_0 (if $e \in E - E_0$). Clearly, $\psi(\lambda) = 1$. In order to describe $\psi(e)$ for $e \neq \lambda$, the function S': E$\longrightarrow \mathcal{P}$(E) is first defined as follows:

$$S'(e) = \{e' \mid (e', e) \in A\} . \tag{3.8}$$

S'(e) is then the set of all entities e' such that $e \in S(e')$. For each entity $e \neq \lambda$, e is displayed once each time that each entity $e' \in S'(e)$ is displayed. The function ψ may then be described as follows:

$$\psi(e) = \begin{cases} 1, \text{ if } e = \lambda, \text{ or} \\ \sum_{e' \in S'(e)} \psi(e'), \text{ otherwise} \end{cases} \tag{3.9}$$

Figure 3.5 shows the graph of a relation A, together with values of ψ.

With the function ψ defined, the number of times that Step 2 of the algorithm is performed may be expressed. This step is performed once each time that an entity $e' \in S(e)$ is identified, where e is an entity which has been identified previously. The number of times that a particular entity e' is identified as being in S(e) is the number of times that e is displayed, i.e., $\psi(e)$. The total number of times that Step 2 is performed is then

$$\sum_{e \in E} \sum_{e' \in S(e)} \psi(e) = \sum_{(e,e') \in A} \psi(e)$$

$$= \sum_{e' \in E - \{\lambda\}} \sum_{e \in S'(e')} \psi(e)$$

$$= \sum_{e' \in E - \{\lambda\}} \psi(e') \quad . \tag{3.10}$$

For the relation A shown in Figure 3.5, the total number of times that Step 2 is performed is

$$\sum_{i=1}^{11} \psi(e_i) = 19 \quad .$$

By using the function ψ, the total time which is consumed performing Step 2 of the algorithm may be expressed. The time which is required to perform this step once is

$$g_2(e,e') = g_s(e,e') + g_x(\rho(e,e')) + g_m(\rho(e,e')) + g_b(D[e']). \tag{3.11}$$

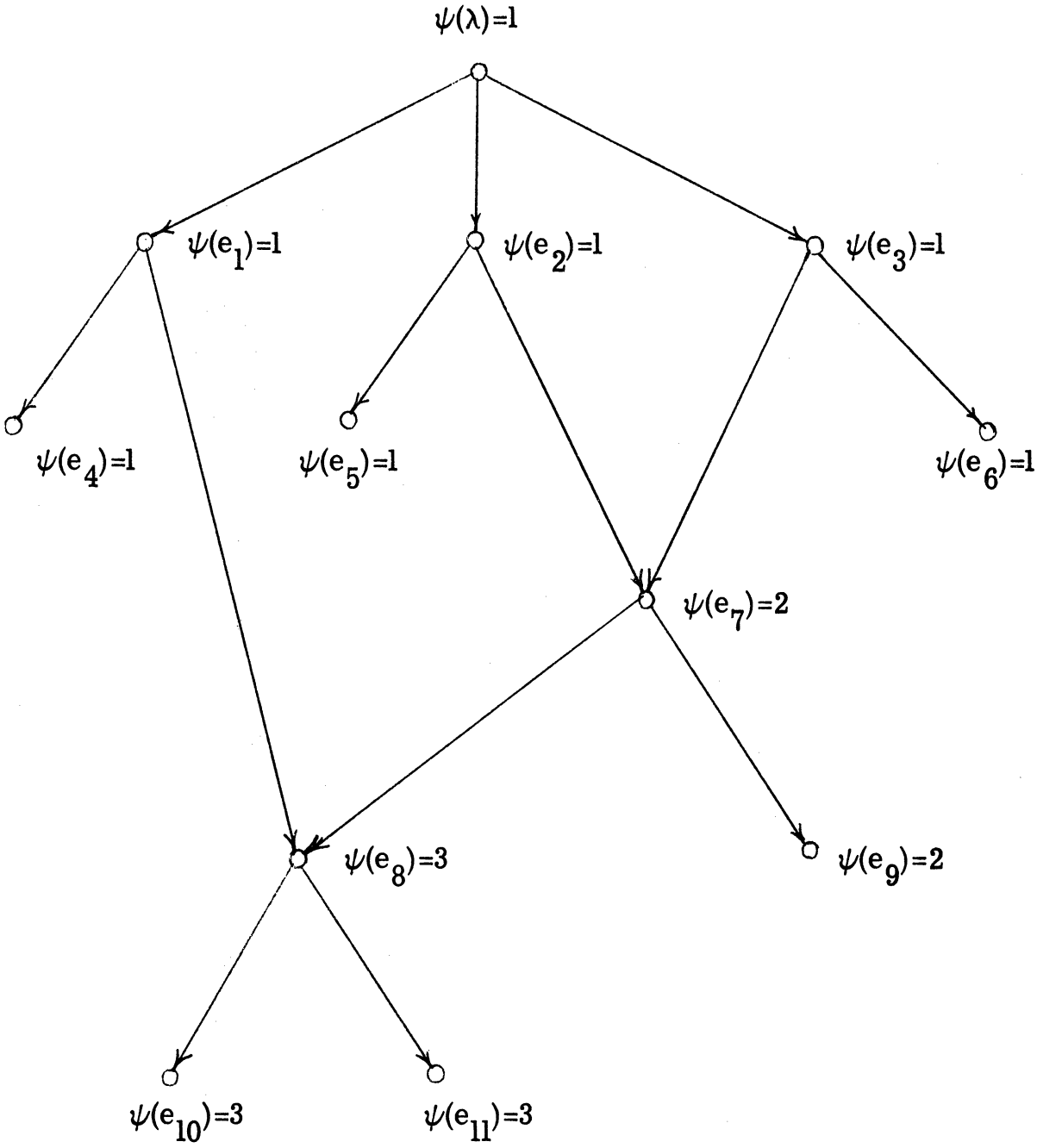


Figure 3.5

A Relation A With Values of ψ Indicated

where

$g_s(e,e')$ is the time which is required to save $b = \rho(e,e')$ and, possibly, $J = H(e)$ on the push-down stack,

$g_x(\rho(e,e'))$ is the time which is required to set X to the second component $D[e']$ of $b = \rho(e,e')$,

$g_m(\rho(e,e'))$ is the time which is required to premultiply $J = H(e)$ by $F'(\rho(e,e'))$ or $G'(\rho(e,e'))$, and

$g_b(D[e'])$ is the time which is required to let b be the element of B such that $\mu(b) = X = D[e']$. $g_b(D[e']) = 0$ if $D[e'] \in E_0/D_0$.

The total time which is consumed performing Step 2 of the algorithm is then

$$\sum_{(e,e')\in A} \psi(e) g_2(e,e') = (\bar{g}_s + \bar{g}_x + \bar{g}_m + \bar{g}_b) \sum_{e\in E-\{\lambda\}} \psi(e) \tag{3.12}$$

where

$$\bar{g}_s = \frac{\sum_{(e,e')\in A} \psi(e) g_s(e,e')}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.13}$$

is the average time which is required to save information pertiment to an element of A on the push-down stack,

$$\bar{g}_x = \frac{\sum_{(e,e')\in A} \psi(e) g_x(\rho(e,e'))}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.14}$$

is the average time which is required to determine the second component of an element of B,

$$\bar{g}_m = \frac{\sum_{(e,e')\in A} \psi(e)\, g_m(\rho(e,e'))}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.15}$$

is the average time which is required to premultiply J by a value of F' or G', and

$$\bar{g}_b = \frac{\sum_{(e,e')\in A} \psi(e) g_b(D[e']}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.16}$$

is the average time which is required to identify the first element of B which describes a subpicture.

Saving the matrix J on the push-down stack is useful only for the structure $\{B, E/D, F'\}$, since this action avoids a matrix multiplication in Step 4 only for this structure. If this matrix is saved, $g_s(e,e')$ may depend on its complexity. For example, in order to reduce the average time which is required for this operation, only the elements of J which differ from the corresponding elements of the identity matrix may be saved. However, if only references to elements of B are saved on the push-down stack, $g_s(e,e')$ is generally constant for all $(e,e') \in A$.

The time $g_x(\rho(e,e'))$ which is required to determine the second component of $\rho(e,e')$ is generally constant for all $\rho(e,e') \in B$ if the representation of each $\rho(e,e')$ contains an explicit reference to $D[e']$. However, if this explicit reference is not included, $g_x(\rho(e,e'))$ may assume different values for various elements $\rho(e,e') \in B$. For example, if no reference to $D[e']$ is stored in the block which represents $\rho(e,e')$, but the elements of each set $P(D[e'])$ are linked together in a ring with $D[e']$, the picture generator must sequence through part of this ring to identify $D[e']$.

The time $g_m(\rho(e,e'))$ which is required to premultiply J by $F'(\rho(e,e'))$ or $G'(\rho(e,e'))$ may differ for various elements $\rho(e,e') \in B$. For example, if $F'(\rho(e,e'))$ is the identity matrix, none of its elements need be stored, and the matrix multiplication may be avoided. However, there generally does exist an element $\rho(e'',e''') \in B$ such that $F'(\rho(e'',e'''))$ is not the identity matrix, and, consequently, some time is required to premultiply J by $F'(\rho(e'',e'''))$. As has been mentioned above, the matrix $G(D[e])^{-1}$ is stored only for $D[e] \in E_0/D_0$. Consequently, if $G'(\rho(e,e')) = G(D[e'])^{-1}$, it must be determined by repeated application of Equation 3.5 for $D[e'] \notin E_0/D_0$. The time which is required for this process is considered to be included in $g_m(\rho(e,e'))$. An algorithm for computing $G(D[e'])^{-1}$ in this way is given below. For each matrix $G(D[e'])^{-1}$ which is determined by this algorithm, the only other matrices $G(D[e'])^{-1}$ which are assumed to be known are those for which $D[e'] \in E_0/D_0$.

Algorithm 3.2

Procedure for determining $G(D[\ e]\ )^{-1}$. (The matrix J in this algorithm is not the matrix J used above in Algorithm 3.1.)

(1) Let J = I, let X = D[e] , and empty a push-down stack for use in this algorithm.

(2) If there exists an element $b' \in B$ such that $\mu(b') = X$, let $b = b'$. Otherwise, let $K = G(X)^{-1}$ and proceed with Step 5.

(3) If $G'(b) \neq G''(b)$, proceed with Step 4. Otherwise, save b and J on the push-down stack. Let X be the second component of b and let J = I. Proceed with Step 2.

(4) If $\mu'(b) \in B$, let $b = \mu'(b)$ and proceed with Step 3.

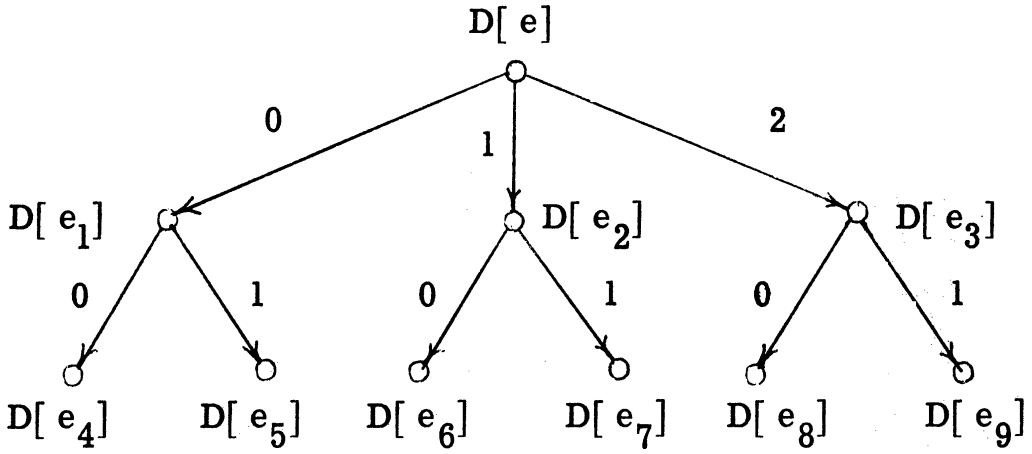
(5) If the push-down stack is empty, terminate with $K = G(D[\ e]\ )^{-1}$. Otherwise, restore b and J from the push-down stack. If $G'(b) = I$, let $J = J\ K$; otherwise, let $J = J\ K^{-1}$. Let K = J and proceed with Step 4.

In order to illustrate this algorithm, a part of a structure $\{B, E/D, \mu, \gamma\}$ is depicted in Figure 3.6, and the steps of Algorithm 3.2 which are needed to determine $G(D[\ e]\ )^{-1}$ from this structure are shown below. Each step is represented below by its number and the items which are affected by performing it.

(1) J = I

X = D[e]

(2) $b = \rho(e, e_1)$



$\mu(\rho(e_1, e_5)) = \rho(e_1, e_4)$ $\quad \mu(\rho(e_2, e_7)) = \rho(e_2, e_6)$ $\quad \mu(\rho(e_3, e_9)) = \rho(e_3, e_8)$

$\mu(\rho(e_1, e_4)) = D[\, e_1]$ $\quad \mu(\rho(e_2, e_6)) = D[\, e_2]$ $\quad \mu(\rho(e_3, e_8)) = D[\, e_3]$

$\mu(\rho(e, e_1)) = D[\, e]$ $\quad \mu(\rho(e, e_2)) = \rho(e, e_1)$ $\quad \mu(\rho(e, e_3)) = \rho(e_1, e_2)$

$\gamma(\rho(e, e_1)) = (I, G(D[\, e_1]), I)$

$\gamma(\rho(e_1, e_4)) = (I, G(D[\, e_4]), I)$

$\gamma(\rho(e_1, e_5)) = (G(D[\, e_5])^{-1},\ G(D[\, e_5]), G(D[\, e_5])^{-1})$

$\gamma(\rho(e, e_2)) = (G(D[\, e_2])^{-1},\ G(D[\, e_2]), I)$

$\gamma(\rho(e, e_3)) = (G(D[\, e_3])^{-1},\ G(D[\, e_3]),\ G(D[\, e_3])^{-1})$

$\gamma(\rho(e_3, e_8)) = (I, G(D[\, e_8]), I)$

$\gamma(\rho(e_3, e_9)) = (G(D[\, e_9])^{-1},\ G(D[\, e_9]),\ G(D[\, e_9])^{-1})$

Figure 3.6

Part of a Structure $\{B, E/D, \mu, \gamma\}$

(3) Stack contains $\rho(e, e_1)$, I

$X = D[e_1]$

$J = I$

(2) $b = \rho(e_1, e_4)$

(3) Stack contains $\rho(e, e_1)$, I, $\rho(e_1, e_4)$, I

$X = D[e_4]$

$J = I$

(2) $K = G(D[e_4])^{-1}$

(5) $b = \rho(e_1, e_4)$; $J = I$; Stack contains $\rho(e, e_1)$, I

$J = G(D[e_4])^{-1}$

$K = G(D[e_4])^{-1}$

(4) $b = \rho(e_1, e_5)$

(3) Stack contains $\rho(e, e_1)$, I, $\rho(e_1, e_5)$, $G(D[e_4])^{-1}$

$X = D[e_5]$

$J = I$

(2) $K = G(D[e_5])^{-1}$

(5) $b = \rho(e_1, e_5)$; $J = G(D[e_4])^{-1}$; Stack contains $\rho(e, e_1)$, I

$J = G(D[e_4])^{-1} G(D[e_5])$

$K = G(D[e_4])^{-1} G(D[e_5])$

(4) $b = \rho(e, e_2)$

(4) $b = \rho(e, e_3)$

(3) Stack contains $\rho(e, e_3)$, $G(D[e_4])^{-1}G(D[e_5])$

$X = D[e_3]$

$J = I$

(2) $b = \rho(e_3, e_8)$

(3) Stack contains $\rho(e, e_3)$, $G(D[e_4])^{-1}G(D[e_5]), \rho(e_3, e_8)$, I

$X = D[e_8]$

$J = I$

(2) $K = G(D[e_8])^{-1}$

(5) $b = \rho(e_3, e_8)$; $J = I$; Stack contains $\rho(e, e_3), G(D[e_4])^{-1}G(D[e_5])$

$J = G(D[e_8])^{-1}$

$K = G(D[e_8])^{-1}$

(4) $b = \rho(e_3, e_9)$

(3) Stack contains $\rho(e, e_3)$, $G(D[e_4])^{-1}G(D[e_5]), \rho(e_3, e_9), G(D[e_8])^{-}$

$X = D[e_9]$

$J = I$

(2) $K = G(D[e_9])^{-1}$

(5) $b = \rho(e_3, e_9)$; $J = G(D[e_8])^{-1}$; Stack contains $\rho(e, e_3), G(D[e_4])^{-1}$
$G(D[e_5])$

$J = G(D[e_8])^{-1}G(D[e_9])$

$K = G(D[e_8])^{-1}G(D[e_9])$

(5) $b = \rho(e, e_3)$; $J = G(D[e_4])^{-1}G(D[e_5])$; Stack empty

$J = G(D[e_4])^{-1}G(D[e_5])G(D[e_9])^{-1}G(D[e_8])$

$K = G(D[e_4])^{-1}G(D[e_5])G(D[e_9])^{-1}G(D[e_8])$

The subset of $B \cup E/D$ which is examined by this algorithm is difficult to describe for the general application of the topological structure. Consequently, a general expression for the time which is required to perform this algorithm is difficult to derive. However, if the form of the topological structure is known for a particular application, the subset of $B \cup E/D$ which is examined by the algorithm is much more obvious. For this reason, the time which is required to perform this algorithm is more easily estimated independently for each application of the structure. Examples for which this time is estimated are given in Chapter 5.

The time $g_b(D[e'])$ which is required to identify the element $b \in B$ such that $\mu(b) = D[e']$ is determined by whether or not $D[e'] \in E_0/D_0$. If $D[e'] \in E_0/D_0$, $g_b(D[e'])$ is the time which is required to determine that $D[e'] \in E_0/D_0$. If $D[e'] \notin E_0/D_0$, $g_b(D[e'])$ generally is the time which is required to determine that $D[e'] \notin E_0/D_0$ and to either examine a pointer or transfer a jump command to the display processor.

3.2.2.3 Displaying Entities in E_0

The time which is required to perform Step 3 of Algorithm 3.1 for the entity $e \in E_0$ is

$$g_3(e) = g_c(e) + g_t(e) , \qquad (3.17)$$

where

$g_c(e)$ is the time which is required for the computer to produce the entity e in a form which is acceptable to the display processor, and

$g_t(e)$ is the time which is required to transfer information which describes e to the display processor.

One of the design decisions which will be discussed in Chapter 4 is whether the display processor or the computer is to form each entity $e \in E_0$ from H(e) and $\beta(e)$. If each entity $e \in E_0$ is formed by the display processor, $g_c(e) = 0$, and $g_t(e)$ is the time which is required to transfer $\beta(e)$ to the display processor. For this case, g(e) is generally proportional to the storage which $\beta(e)$ occupies. However, if e is formed from H(e) and $\beta(e)$ by the computer, $g_c(e) > 0$, and $g_t(e)$ is the time which is required to transfer e to the display processor. For this case, $g_c(e)$ and $g_t(e)$ may not be proportional to the storage which $\beta(e)$ occupies. For example, $\beta(e)$ may be stored as a mixture of character codes and vectors. Each character code may require more time than a vector to convert into a format which is acceptable to the display processor, and much more information generally must be transferred to the display processor in order to describe it. Furthermore, more character codes than vectors may generally be stored per storage location. The total time which is consumed displaying entities in E_0 is then

is the average time which is required to restore information relevant to an element of A from the push-down stack,

$$\bar{g}_h = \frac{\sum_{(e,e')\in A} \psi(e)\, g_h(\rho(e,e'))}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.24}$$

is the average time which is required to either premultiply J by a value of G'' or to invert a value of F' and premultiply J by the resulting matrix, and

$$\bar{g}_n = \frac{\sum_{(e,e')\in A} \psi(e)\, g_n(\rho(e,e'))}{\sum_{e\in E-\{\lambda\}} \psi(e)} \tag{3.25}$$

is the average time which is required to examine a value of μ',

3.2.2.5 Summary

From Equations 3.12, 3.18, and 3.22, the total time g which is required to perform Algorithm 3.1 is

$$g = \sum_{(e,e')\in A} \psi(e) g_2(e,e') + \sum_{e\in E_0} \psi(e) g_3(e) + \sum_{(e,e')\in A} \psi(e) g_4(e,e')$$

$$= (\bar{g}_s + \bar{g}_r + \bar{g}_m + \bar{g}_h + \bar{g}_b + \bar{g}_n + \bar{g}_x) \sum_{e\in E-\{\lambda\}} \psi(e) + (\bar{g}_c + \bar{g}_t) \sum_{e\in E_0} \psi(e) . \tag{3.26}$$

If there exist no distinct entities e, e', and e'' such that $e \in S(e') \cap S(e'')$, $\psi(E) = \{1\}$, and Equation 3.26 reduces to the following simpler form:

$$g = (|E| - 1)(\bar{g}_s + \bar{g}_r + \bar{g}_m + \bar{g}_h + \bar{g}_b + \bar{g}_n + \bar{g}_x) + |E_0|(\bar{g}_c + \bar{g}_t) \qquad (3.27)$$

3.3 Average Response Time

The cost of an implementation of the topological structure is the average time which is required to modify or interpret the structure in response to a control language input. This cost will be denoted by $\bar{r}$ and may be expressed as follows:

$$\bar{r} = \sum_{k' \in K} p(k')\, \bar{r}_k(k'), \qquad (3.28)$$

where

- K is the set of possible control language inputs,
- p(k') is the probability that the next control language input will be k', and
- $\bar{r}_k(k')$ is the average time which is required to respond to input k'.

In further discussion, each response will be assumed to be performed in sequence, i.e., if an input k_2 is received while the response to the input k_1 is in progress, the response to k_1 is completed before the response to k_2 is begun. If this were not true, the initiation of the response to k_2 could interfere with the completion of the response to k_1. For example, the first step in responding to k_2 might be to destroy a portion of the structure which is needed to complete the response to k_1.

The response to any control language input will be considered to be a sequence of operations which are performed on the topological structure. Although the response to an input often includes other operations such as communication with another computer, numerical computation, etc., these operations will not be considered when computing $\bar{r}$ because they are independent of the way in which the topological structure is implemented. The average time $\bar{r}_k(k')$ which is required to respond to the input k' may then be expressed as follows:

$$\bar{r}_k(k') = \sum_{\ell' \epsilon L} \bar{n}(k', \ell') \bar{r}_\ell(\ell'), \tag{3.29}$$

where

L is the set of operations which may be performed on the structure,

$\bar{n}(k', \ell')$ is the average number of times that operation ℓ' must be performed in order to respond to the input k', and

$\bar{r}_\ell(\ell')$ is the average time which is required to perform operation ℓ'.

From Equations 3.28 and 3.29,

$$\begin{aligned} \bar{r} &= \sum_{k' \epsilon K} p(k') \sum_{\ell' \epsilon L} \bar{n}(k', \ell') \bar{r}_\ell(\ell') \\ &= \sum_{\ell' \epsilon L} \sum_{k'' \epsilon K} p(k') \bar{n}(k', \ell') \bar{r}_\ell(\ell') \\ &= \sum_{\ell' \epsilon L} f(\ell') \bar{r}_\ell(\ell') , \end{aligned} \tag{3.30}$$

where

$$f(\ell') = \sum_{k' \in K} p(k')\bar{n}(k', \ell') \qquad (3.31)$$

is the average number of times that operation ℓ' is performed in order to respond to a control language input.

The set L may be chosen arbitrarily. However, certain basic operations may be applied to perform all modifications and interpretations of the structure. Although these operations may not all be available in some implementations, the operations which are available may be described as sequences of these basic operations. For this reason, L will be considered to be the set of basic operations in all further discussion.

3.3.1 Definition of Basic Operations

There are two types of basic operations: (1) those which modify the structure, and (2) those which interrogate the structure. The basic operations which modify the structure are the following:

ℓ_1: Create an element of E/D.

ℓ_2: Destroy an element of E/D

ℓ_3: Create an element $\rho(e,e') \in B$, given $D[e]$, $D[e']$, $\mu(\rho(e,e'))$, and either $F'(\rho(e,e'))$ or $\gamma(\rho(e,e'))$.

ℓ_4: Destroy an element $\rho(e,e') \in B$.

Sequences of these operations may be applied to perform any modification of the topological structure, since this structure is described

by B, E/D, and either F' or μ and γ. However, no attempt has been made to define operations which modify subpictures which are used to generate entities in E_0, for such operations would affect only the appearance of the picture, but not its structure.

The basic operations which interrogate the structure may be grouped into two classes: those which examine elements of E/D, and those which examine elements of B. The operations which examine elements of E/D are the following. For each operation, the class D [e] is assumed to be given.

ℓ_5: Identify the elements of P(D[e]).

ℓ_6: Identify $\rho(e,e')$ such that $\mu(\rho(e,e')) = D[e]$.

(Applicable only if $D[e] \notin E_0/D_0$.)

ℓ_7: Identify $\rho(e,e')$ such that $\mu'(\rho(e,e')) = D[e]$.

(Applicable only if $D[e] \notin E_0/D_0$.)

ℓ_8: Retrieve $G(D[e])^{-1}$.

(Applicable only to the structure $\{B, E/D, \mu, \gamma\}$ for $D[e] \in E_0/D_0$.)

These operations are sufficient to obtain from D[e] all of the information which might be stored to represent it in memory.

The operations which examine elements of B are the following. For each operation, the element $\rho(e,e') \in B$ is assumed to be given.

ℓ_9 : Identify $\mu(\rho(e,e'))$.

ℓ_{10}: Identify $\mu'(\rho(e,e'))$.

ℓ_{11}: Identify $D[e]$.

ℓ_{12}: Identify $D[e']$.

ℓ_{13}: Determine $F'(\rho(e,e'))$.

(Applicable only to the structure $\{B,E/D,F'\}$.)

ℓ_{14}: Determine the form of $\gamma(\rho(e,e'))$ (i.e., whether $\gamma(\rho(e,e')) = (I,G(D[e'],I),(I,G(D[e']),G(D[e'])^{-1}),$ $(G(D[e'])^{-1},G(D[e']),I),$ or $(G(D[e'])^{-1},G(D[e']),G(D[e'])^{-1})$.

These operations are sufficient to obtain from $\rho(e,e')$ all of the information which might be stored to represent it in memory. Operations ℓ_5 through ℓ_{14} may then be applied to obtain from the structure all of the information which it represents, except the subpictures which are needed to generate entities in E_0.

Although the operations ℓ_1, ℓ_2, ℓ_3, and ℓ_4 are sufficient to modify the structure, they may not be suitable for certain modifications of the structure $\{B,E/D,\mu,\gamma\}$. In particular, if an element $\rho(e,e') \in B$ in this structure is modified by first destroying it (via ℓ_4) and then re-creating it in a different form (via ℓ_3), some entities may be displayed incorrectly. These entities are incorrectly displayed if $\rho(e,e')$ is destroyed and not replaced with a new element until after the subpicture $\beta(e)$ has again been transformed to produce an entity $e'' \in D[e]$. If $G(D[e']) \neq I$ and $\gamma(\rho(e,e'))$ is either $(I,G(D[e']),I)$ or

$(G(D[e'])^{-1}, G(D[e']), G(D[e'])^{-1})$ before $\rho(e,e')$ is destroyed, subpictures are concatenated incorrectly to form e''. Furthermore, the entities which are produced by concatenating subpictures to $\beta(e)$ may be displayed as incorrect transformations of these subpictures. Not only is this effect undesirable in that it produces an incorrect picture, but it may also cause demonstrative inputs to be interpreted incorrectly, since some entities are momentarily displayed at incorrect coordinates.

One common modification of an element of B in the structure $\{B, E/D, \mu, \gamma\}$ is the replacement of the second component $D[e']$ of $\rho(e,e')$ with a different class $D[e'']$. In particular, the ability to modify an element of B in this way is needed if entities are to be moved independently, since entities may be moved independently only by substituting one class of artificial entities for another. In order to provide this facility, the following additional basic operation is defined for modifying elements of B in the structure $\{B, E/D, \mu, \gamma\}$.

ℓ_{15}: Replace $\rho(e,e')$ with $\rho(e,e'')$ such that $\mu(\rho(e,e'))=\mu(\rho(e,e''))$, $\mu'(\rho(e,e'))=\mu'(\rho(e,e''))$, and the form of $\gamma(\rho(e,e''))$ is the same as that of $\gamma(\rho(e,e'))$.

This operation is equivalent to destroying an element of B (via ℓ_4) and then recreating it in a different form (via ℓ_3). However, since the entire change in the element is accomplished by one operation, the difficulty mentioned above cannot occur.

3.3.2 Time Required to Perform Basic Operations

The time which is required to perform some of the basic operations depends on which references to elements of $B \cup E/D$ are stored to represent the structure. The time which is required to perform many of the operations which retrieve information from the structure ($\ell_5, \ell_6, \ell_7, \ell_8, \ell_{10}, \ell_{11}$, and ℓ_{12}) is minimized if all of the references to these elements which were described in Section 3.1 are stored. However, the operations ℓ_3 and ℓ_4, which create and destroy elements of B, may require more time to perform if all of these references are stored. Consequently, even when a large amount of storage is available, the average response time $\bar{r}$ is not necessarily minimized by storing all of the references to elements of $B \cup E/D$. In some applications, not enough storage is available to store all of these references. In this event, those which are to be omitted should be selected so that $\bar{r}$ is minimized subject to the storage constraint.

Whenever a particular type of reference is omitted from the structure, an operation which would normally examine one of these references to identify an element of $B \cup E/D$ must search a portion of the structure in order to identify that element. This search of the structure may be performed in either of the following two ways:

(1) The program which is operating on the structure may search the structure itself.

(2) The picture generator may identify the appropriate element of $B \cup E/D$ to the program which is operating on the structure as this element is encountered in the course of generating the picture.

Generally, the former method has been used when the picture generator has been a program, whereas the latter method has been used when the display processor hardware has been used as the picture generator. However, the method to be used can be chosen independently of the way in which the picture generator is implemented.

If, in order to perform a basic operation, a portion of the structure is searched by the program which is operating on it, the procedure which is used may be much simpler than Algorithm 3.1. For example, if ℓ_7 is applied to the class D[e] in the second list structure described in Section 3.1.4, the only elements of B which must be examined are those whose first component is D[e]. However, if Algorithm 3.1 is used, all elements of B which are components of paths from e to entities in E_0 and all elements $D[e'] \in E/D$ such that there exists a path from e to e' must be examined. Consequently, the average time which is required to perform a basic operation which requires a search is often larger if the picture generator, rather than the program which is operating on the structure, performs the search. However, if the display processor is used as the picture generator, the total time during which the computer is devoted to searching the structure is generally smaller if the display processor identifies the appropriate elements of $B \cup E/D$ to the program.

Like those operations which may wait for the picture generator to identify certain elements of $B \cup E/D$, the operations which modify elements of B must often wait for the picture generator to finish Algorithm 3.1, as will be discussed below in Section 3.3.2.1. The average time $\bar{r}_{\ell}(\ell)$ which is required to perform Operation ℓ may then be expressed as the sum of the average time which elapses while the program which is operating on the structure waits for the picture generator and the average time which elapses while this program performs the remainder of the operation. Whenever the program is not waiting for the picture generator, the computer generally must contribute both to the execution of the operation and the generation of the picture. The average time $\bar{r}_{\ell}(\ell)$ may then be expressed as follows:

$$\bar{r}_{\ell}(\ell) = \bar{r}_{m}(\ell) + (\bar{n}_{f}(\ell) - \bar{n}_{w}(\ell))g + \bar{n}_{w}(\ell)t_{f}, \qquad (3.32)$$

where

$\bar{r}_{m}(\ell)$ is the average time which the program which is operating on the structure devotes to the execution of instructions which contribute to Operation ℓ,

$\bar{n}_{f}(\ell)$ is the average total number of times that the picture is produced while Operation ℓ is performed,

$\bar{n}_{w}(\ell)$ is the average number of times that the picture is produced while the program which is operating on the structure is waiting for the picture generator, and

t_{f} is the time which elapses while the picture is generated once.

The second term in this equation represents the average amount of time that the execution of instructions which contribute to Operation ℓ is suspended while the computer is devoted to the generation of the picture. The last term represents the average amount of time that the program which is modifying the structure must wait for the picture generator either to identify an element of $B \cup E/D$ or to complete Algorithm 3.1.*

The quantities $\bar{r}_m(\ell)$ and $\bar{n}_w(\ell)$ are determined by the way in which Operation ℓ is implemented. The average number of times $\bar{n}_f(\ell)$ that Algorithm 3.1 is performed while Operation ℓ is being performed, however, is also determined by the manner in which the display is refreshed. Since the picture must be continually generated, $\bar{r}_\ell(\ell)$ may also be expressed as follows:

$$\bar{r}_\ell(\ell) = \bar{n}_f(\ell) t_f \, . \tag{3.33}$$

By considering both this equation and Equation 3.32, $\bar{n}_f(\ell)$ may be eliminated to yield

$$\bar{r}_\ell(\ell) = \bar{r}_m(\ell) t_f / (t_f - g) + \bar{n}_w(\ell) t_f \tag{3.34}$$

If $\bar{r}_w(\ell)$ is defined to be the average time which the program which is operating on the structure must wait for the picture generator to reach a particular part of the structure so that the Operation ℓ may be completed, this equation may be written as follows:

*Whenever $\bar{n}_w(\ell) > 0$, the picture generator must stop generating the picture very briefly in order to either identify an element of $B \cup E/D$ to the program which is modifying the structure or allow this program to modify references to elements of B. The time which is required to execute computer instructions while the picture generator is stopped is much less than t_f, and, consequently, it does not appear in this equation. However, this time will not be ignored later when Operations ℓ_3 and ℓ_4 are discussed.

$$\bar{r}_{\ell}(\ell) = \bar{r}_{m}(\ell)\, t_f/(t_f - g) + \bar{r}_{w}(\ell) \,. \qquad (3.35)$$

The factor $t_f/(t_f - g)$ may be interpreted to be a degradation in response which results from the fact that the computer must contribute both to the performance of Operation ℓ and to the generation of the picture. In general, then, this factor will appear in each equation which describes the average time which is required to perform a basic operation as a factor in a term which represents the time during which the computer executes instructions which contribute to the operation while the picture is being generated.

3.3.2.1 Time Required to Modify the Structure

The basic operations which modify the structure are $\ell_1, \ell_2, \ell_3, \ell_4$ and ℓ_{15}. Operations ℓ_1 and ℓ_2, respectively, create and destroy elements of E/D, whereas Operations ℓ_2 and ℓ_4, respectively, create and destroy elemen of B. Operation ℓ_{15} was defined to modify an existing element of B in the structure $\{B, E/D, \mu, \gamma\}$. The time which is required to perform each of these basic operations is discussed below.

3.3.2.1.1 Creating and Destroying Elements of E/D

The time which is required to create and destroy elements of E/D (via ℓ_1 and ℓ_2) depends chiefly on the storage allocation algorithm which is used, rather than on the way in which the structure is implemented. Creating an element of E/D involves creating a storage block and copying information into it, whereas destroying an element of E/D involves destroying a storage block. (Operation ℓ_2 is assumed to be applicable to the class D[e] only after all elements of B whose first or second component is D[e] have been destroyed.) Consequently, the average time which is required to create an element of E/D is

$$\bar{r}_{\ell}(\ell_1) = (\bar{r}_g + r_c(|E_0/D_0|\bar{s}_{d0} + |E/D - E_0/D_0|s_d)/|E/D|)t_f/(t_f - g), \quad (3.36)$$

where

$\bar{r}_g$ is the average time which is required to create a storage block, and

r_c is the time which is required to copy the contents of one storage location into another.

Similarly, if $\bar{r}_d$ denotes the average time which is required to destroy a storage block, the average time which is required to destroy an element of E/D is

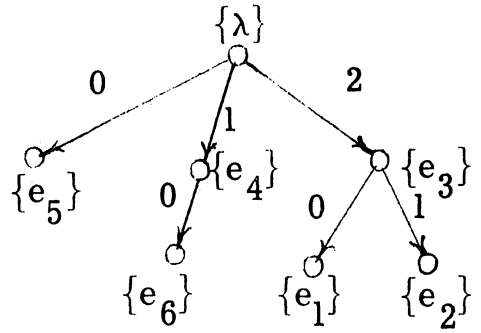
$$\bar{r}_{\ell}(\ell_2) = \bar{r}_d t_f/(t_f - g) . \quad (3.37)$$

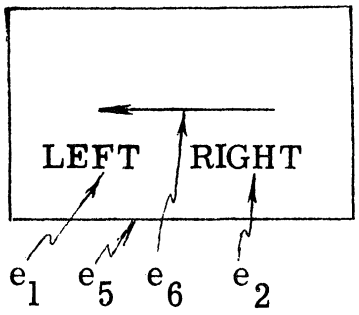
3.3.2.1.2 Creating Elements of B

The time which is required to create and destroy elements of B (via ℓ_3 and ℓ_4), however, is much more dependent on the way in which the structure is implemented. The process of creating an element $b \in B$ involves creating a storage block, copying information into this storage block, and inserting this block into the existing structure. If references to values of μ are not stored in the structure, the process of inserting the block into the structure involves storing a reference to this block in the block which is to represent $\mu(b)$. However, if values of μ are stored in the structure, a reference to this block must be stored in both the block which is to represent $\mu(b)$ and the block which is to represent $\mu'(b)$. In either case, the element of

$B \cup E/D$ which is to be $\mu'(b)$ must be identified, for a reference to it must be stored in the block which is created to represent b. This element is easily identified, however, for a reference to it is stored in the block which represents $\mu(b)$.

The process of creating an element of B is complicated by the fact that the picture generator and the program which is modifying the structure must often be synchronized in order to store references to the newly created element in the structure. For example, Figure 3.7 illustrates some of the difficulties which occur if this synchronization is not performed when modifying the structure $\{B, E/D, \mu, \gamma\}$. Figure 3.7a shows a picture which contains the two light buttons e_1 and e_2 and a border e_5. In addition, the set of light buttons e_3 is represented in the structure, and the entity e_4, which represents a set of arrows which point left and right, is also represented. The latter set contains the arrow e_6 in Figure 3.7a. The purpose of the program which operates on this picture is to allow the user to create arrows by referring to e_1 and e_2 with a light pen. If he selects e_1, an arrow which points to the left is created, whereas, if he selects e_2, an arrow which points to the right is created. Figure 3.7b shows the form which the picture should assume if the user selects e_1. ($G(\{e_6\})$ is assumed to represent translation.) Figure 3.7c, however, shows the form which the picture might assume if the picture generator and the program which modifies the structure are not synchronized. This distorted picture is produced if a reference to $\rho(e_4, e_7)$ is stored in





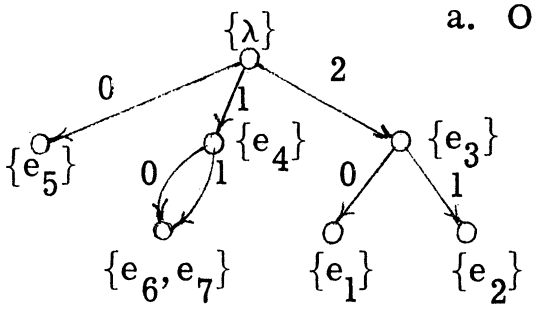
$\mu(\rho(\lambda, e_3))=\rho(\lambda, e_4)$

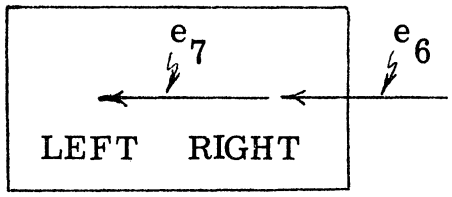
$\mu(\rho(\lambda, e_4))=\rho(\lambda, e_5)$

$\gamma(\rho(e_4, e_6))=(I, G(\{e_6\}), I)$

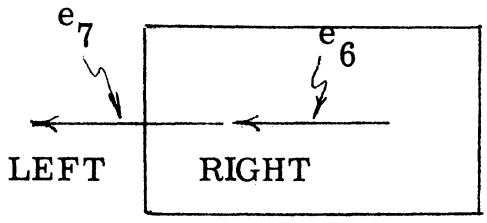
$\gamma(\rho(\lambda, e_4))=(G(\{e_4\})^{-1}, G(\{e_4\}), I)$

a. Original Picture





b. Modified Picture



c. Distorted Picture

Figure 3.7

Example of Picture Distortion Due to Lack of Synchronization

the block which represents $\{e_4\}$ while the entity e_6 is being plotted and before $\mu'(\rho(e_4, e_6))$ has been determined by the picture generator. The picture generator then proceeds to display e_7, e_1, and e_2. However, since $G'(\rho(\lambda, e_4)) = G(\{e_4\})^{-1}$ was modified just after it was used in the picture generation process, these entities are not displayed at the proper coordinates. Since the picture is assumed to be continually generated, the picture is soon restored to its correct form shown in Figure 3.7b. However, the momentary distortion of the picture may move e_2 under the light pen and cause another arrow to be created, contrary to the intentions of the user.

If the picture is not to be distorted when an element of B is added to the structure $\{B, E/D, \mu, \gamma\}$, the references to the newly created element of B must be stored in the structure between generations of the picture. The program which is modifying the structure, then, must either wait until the picture generator has completed Algorithm 3.1 and store references to the newly created element of B before a picture is again generated from the structure, or it must interrupt the picture generator, store references to this element, and then restart the picture generator with the first step of Algorithm 3.1. The former procedure is time-consuming, and, consequently, may not be suitable if many elements of B are to be created in response to a single input. The latter procedure provides for much more rapid creation of elements of B. However, the picture may momentarily disappear if a large number of elements of B are created in rapid succession.

Because the picture generation process is assumed to have priority over the process of responding to control language inputs, a program which creates a new element of B for the structure $\{B, E/D, \mu, \gamma\}$ will be assumed to wait for the picture generator to complete Algorithm 3.1 before storing references to the newly created element. The time at which ℓ_3 is initiated is generally independent of the picture generation process. Consequently, when the program which is modifying the structure is ready to store references to a newly created element of B, it is equally likely that the amount of time which is needed for the picture generator to complete Algorithm 3.1 is any value between 0 and t_f. The average time which the program which is modifying the structure must wait is then $t_f/2$.

Even if the topological structure is $\{B, E/D, F'\}$, synchronization of the program which is modifying the structure with the picture generator may be necessary. If a reference to an element of $B \cup E/D$ which occupies more than one storage location is to be modified, the program which is modifying the structure must insure that the picture generator does not interpret this reference when it is only partially modified. This condition is insured if the program which is modifying the structure waits for the picture generator to complete Algorithm 3.1. However, the operation may be completed more rapidly if the program which is modifying the structure interrupts the picture generator, stores references to the newly created element

of B, and then resumes the picture generator from the point in Algorithm 3.1 at which it was interrupted. Although the alternative to interrupt the picture generator was discarded above when discussing the addition of an element of B to the structure $\{B, E/D, \mu, \gamma\}$, the picture generation process is not affected appreciably if the picture generator is resumed, rather than restarted with the first step of Algorithm 3.1. Consequently, the picture generator will be assumed to be interrupted when necessary in order to store references to a newly created element of B for the structure $\{B, E/D, F'\}$. The time which is required to interrupt and resume the picture generator will be assumed to be small and will be ignored.

The average time which is required to synchronize the program which is modifying the structure with the picture generator and to store references to a newly created element of B will be denoted by $\bar{r}_1$. Then, the average time which is required to create an element $b \in B$ may be expressed as follows:

$$\bar{r}_{\ell}(\ell_3) = (\bar{r}_g + r_c \bar{s}_b) t_f/(t_f - g) + \bar{r}_{\ell}(\ell_{10}) + \bar{r}_1 . \qquad (3.38)$$

The first step in creating an element of B is to create a storage block. This process requires the average time $\bar{r}_g$. The average time which is required to copy information into this storage block is $r_c \bar{s}_b$. Because a reference to $\mu'(b)$ must be stored in this block, the term $\bar{r}_{\ell}(\ell_{10})$ is included in this equation to represent the average time which is required to identify the element which is to be $\mu'(b)$ from

the element which is to be $\mu(b)$ (which is specified). Finally, $\bar{r}_1$ is the average time which is required to insert the completed block into the existing structure. ($\bar{r}_1$ is $\bar{r}_w(\ell_3)$ plus the time which is required to store references to a newly created element of B.)

3.3.2.1.3 Destroying Elements of B

A program which performs Operation ℓ_4 must also be synchronized with the picture generator. Whenever an element of B in the structure $\{B, E/D, \mu, \gamma\}$ is to be destroyed, the references to it which appear in the structure must be modified when the picture generator is not operating. Otherwise, the same type of problem as that shown in Figure 3.7 might occur. Consequently, a program which removes an element of B from the structure $\{B, E/D, \mu, \gamma\}$ must wait for the picture generator to complete Algorithm 3.1 before it modifies references to this element. As has been mentioned above, the average time which this program must wait for the picture generator is $t_f/2$.

References to an element $\rho(e, e') \epsilon B$ in the structure $\{B, E/D, F'\}$ must be modified while the picture generator is not displaying e' as a collection of entities in E_0. Otherwise, a reference to $\rho(e, e')$ may later be restored from the push-down stack by Algorithm 3.1 after

$\rho(e,e')$ no longer exists. The necessary synchronization with the picture generator may be accomplished either by waiting for the picture generator to complete Algorithm 3.1 or by waiting until $\rho(e,e')$ is not on the push-down stack via Algorithm 3.3, which is given below.

Algorithm 3.3

Procedure for insuring that $\rho(e,e') \epsilon B$ is not on the push-down stack used by Algorithm 3.1 when references to this element are modified.

(1) Store in the block which represents D[e] an indication to the picture generator to halt after setting X to D[e] in Step 2 of Algorithm 3.1.

(2) Wait until either $\rho(e,e')$ is not on the push-down stack used by Algorithm 3.1 or the picture generator has halted.

(3) Modify references to $\rho(e,e')$.

(4) Remove the indication to the picture generator which was stored in Step 1 and resume the picture generator if it halted. ∎

If the program which is modifying the structure uses Algorithm 3.3 to synchronize with the picture generator, the time which it must wait depends on both t_f and the time which is required to generate the part of the picture which is being modified. In order to discuss this time interval, $t_s(D[e])$ is defined to be the time which is required to display an entity in $D[e]$ as a collection of entities in E_0. Then, if references to an element of B whose first component is $D[e]$ are to be modified and the picture generator is displaying an entity in $D[e]$, the average time which the program which is modifying the structure must wait is $t_s(D[e])/2$. The probability that the picture generator is displaying an entity $e' \in D[e]$ is

$$\frac{t_s(D[e])}{t_f} \sum_{e' \in D[e]} \psi(e') .$$

If the average time which is required to determine that a given element $\rho(e'',e)$ is not on the push-down stack is assumed to be small relative to $t_s(D[e])$, the time which the program which is modifying the structure must wait to synchronize with the picture generator is then

$$\frac{t_s(D[e])}{2} \left(\frac{t_s(D[e])}{t_f} \sum_{e' \in D[e]} \psi(e') \right) .$$

Since $t_s(D[e]) \sum_{e' \in D[e]} \psi(e') < t_f$ for $D[e] \neq \{\lambda\}$, the above quantity is less than or equal to $t_f/2$.

Note, however, that ℓ_4 may be applied to elements of B after they have been removed from the structure. For example, in order to destroy the two rows of polygons in the picture shown in Figure 2.1 (whose set B is shown in Figure 2.10), $\rho(\lambda, e_4)$ and $\rho(\lambda, e_6)$ may first be destroyed. Then, $\rho(e_4, e_1) = \rho(e_6, e_7)$, $\rho(e_4, e_2) = \rho(e_6, e_8)$, and $\rho(e_4, e_3) = \rho(e_6, e_9)$ no longer are part of the structure which is being scanned by the picture generator. These elements may be subsequently destroyed via ℓ_4. The program which destroys these latter elements, however, may modify references to these elements immediately, since the probability that any one of them is on the push-down stack used by Algorithm 3.1 is zero.

As a result, the actual time which is required to execute Algorithm 3.3 is diffuicult to specify. The above discussion has indicated an upper bound for this time through the use of the function t_s. However, values of t_s are also difficult to specify for any class other than $\{\lambda\}$. For simplicity, then, a program which destroys an element of B which appears in either of the structures $\{B, E/D, F'\}$ or $\{B, E/D, \mu, \gamma\}$ subsequently will be assumed to wait for the picture generator to complete Algorithm 3.1 before it modifies references to this element. With this assumption, the average time $t_f/2$ which the program waits for the picture generator is an upper bound for the time which the program would wait if Algorithm 3.3 were used.

The average time which is required to synchronize the program which is modifying the structure with the picture generator and to remove references to an element of B from the structure will be denoted by $\bar{r}_2$ (i.e., $\bar{r}_2$ is $\bar{r}_w(\ell_4)$ plus the time which is required to modify references to the element of B which is to be destroyed). Then, the average time which is required to perform Operation ℓ_4 may be expressed as follows:

$$\bar{r}_\ell(\ell_4) = \bar{r}_d t_f/(t_f - g) + \bar{r}_\ell(\ell_9) + \bar{r}_\ell(\ell_{10}) + \bar{r}_2. \qquad (3.39)$$

The average time which is required to destroy the storage block which represents $\rho(e,e') \in B$ is $\bar{r}_d$. Before this block is destroyed, however, the block which represents $\mu(\rho(e,e')) \in B$ must be identified, for it contains a reference to $\rho(e,e')$ which must be changed. The average time which is required to identify this block is $\bar{r}_\ell(\ell_9)$. The value to which this reference should be changed is $\mu'(\rho(e,e'))$. The average time which is required to determine this value is $\bar{r}_\ell(\ell_{10})$. If references to values of μ are stored, the references to $\rho(e,e')$ in the block which represents $\mu'(\rho(e,e'))$ must also be changed to references to $\mu(\rho(e,e'))$.

3.3.2.1.4 Modifying Elements of B in the Structure $\{B, E/D, \mu, \gamma\}$

Operation ℓ_{15} may be applied to the element $\rho(e,e') \in B$ by changing the value of the reference to D[e'] which is stored in the block which represents $\rho(e,e')$. In order to avoid the type of problem depicted in Figure 3.7, this reference must be changed when the push-down stack used in Algorithm 3.1 is empty. Consequently, the

program which is modifying the structure must wait until the picture generator completes Algorithm 3.1 before it changes this reference. The average time $\bar{r}_{\ell}(\ell_{15})$ which is required to perform this operation is then approximately $t_f/2$. (The time which is required to modify the reference is much less than $t_f/2$ and, therefore, may be ignored.)

3.3.2.2 Time Required to Interrogate the Structure

The remaining basic operations which were defined in Section 3.3.1 interrogate the structure. The procedure which is used to perform each of these basic operations generally is determined by which references to elements of $B \cup E/D$ are stored in the structure and whether or not the display processor performs Algorithm 3.1. The various procedures which will be considered for each operation are discussed below, together with the average time which is required to perform each operation by each procedure.

3.3.2.2.1 Identifying P(D[e]) for a Given Class D[e]

Operation ℓ_5 may be performed in the following three ways:

(1) The ring which links elements of P(D[e]) with D[e] may be examined. (Whether or not this ring exists is determined by the design decision u_{10}, which will be described in Chapter 4.)

(2) The picture generator may identify each element of P(D[e]) to the program which is interpreting the structure as it is encountered in the course of generating the picture.

(3) The structure may be searched by the program which is interpreting it in order to identify the elements of P(D[e]).

The first of these procedures is clearly the fastest, since it does not require that the entire structure be searched by either the picture generator or the program which is interpreting the structure. However, if each set P(D[e]) is not represented by a ring, one of the other two methods must be used. Of the remaining two methods, searching the structure with the program which is interpreting it is generally faster. However, less computation time may be required if the picture generator identifies the elements of P(D[e]). The design decision u_4 (to be discussed in Chapter 4) describes which of these procedures is used if each set P(D[e]) is not represented by a ring.

If each set P(D[e]) is represented by a ring, this ring may be implemented either with forward pointers only or with both forward and backward pointers, as determined by the design decision u_{11}, which will be discussed in Chapter 4. The amount of time which is required to remove a block from the ring is often less if both forward and backward pointers are stored, but the average time which is required to perform ℓ_5 is independent of this feature. The following algorithm may be used to identify the elements of P(D[e]) from this ring:

Algorithm 3.4

Procedure for identifying the elements of P(D[e]) from a ring which represents P(D[e]).

(1) Let $X = \phi$ and let $x = D[e]$.

(2) Let x be the element of $B \cup E/D$ which is referenced by the forward pointer which is stored in the block which represents

x. If x=D[e] , terminate with X=P(D[e]). Otherwise, let $X = X \cup \{x\}$ and repeat this step.

If the initialization step is considered to require a negligible amount of time, the time which is required to perform this algorithm is the total time which is spent performing Step 2 of the algorithm. This step is performed $1 + |P(D[e)|$ times, since the ring which represents P(D[e]) contains $1+ |P(D[e])|$ blocks. Since each element of B can belong to only one set P(D[e]),

$$\sum_{D[e] \in E/D} |P(D[e])| = |B|, \tag{3.40}$$

and the average cardinality of P(D[e]) is $|B|/|E/D|$. Consequently, the average time which is required to perform Algorithm 3.4 is

$$\bar{r}_{\ell}(\ell_5)\Big|_{u_{10}=1} = (T_p+T_c)(1+ |B|/|E/D|)t_f/(t_f-g), \tag{3.41}$$

where

T_p is the time which is required to obtain a pointer from a block in the structure, and

T_c is the time which is required to compare two pointers to elements of $B \cup E/D$ in order to determine whether or not they refer to the same element.

If each set P(D[e]) is not represented by a ring of storage blocks, the entire structure must be searched by either the picture generator or the program which is interpreting the structure in order to perform

ℓ_5. If the program which is interpreting the structure must wait for the picture generator to search the structure, it must first synchronize with the picture generator by waiting for the picture generator to finish Algorithm 3.1. Then, it must wait a time t_f to obtain from the picture generator all elements of a set P(D[e]). The average time which is required to perform ℓ_5 by this procedure is then approximately $3t_f/2$.

If the program which is interpreting the structure is to search the structure in order to perform operation ℓ_5, a simpler procedure than Algorithm 3.1 may be used. Clearly, there is no need to examine the coordinate transformation matrices which appear in the structure. Furthermore, no element of B need be examined more than once, since the purpose of ℓ_5 is to determine a set of elements of B, rather than to determine entities from the structure. Not only does no element of B need to be examined more than once, but those elements of B which are components only of paths from $e' \epsilon D[e]$ to other entities need not be examined at all. An algorithm which the program which is interpreting the structure may use in order to perform operation ℓ_5 is the following:

Algorithm 3.5

Procedure for determining P(D[e]) from a structure in which P(D[e]) is not explictly represented.

(1) Let $S = \{\lambda\}$, $Y = \phi$, $Z = \phi$, let b be the element of B such that $\mu(b) = \{\lambda\}$, and empty a push-down stack for use in this algorithm.

(2) Save b on the push-down stack. Let $X \in E/D$ be the second component of b. If $X = D[e]$, let $Z = Z \cup \{b\}$ and proceed with Step 5.

(3) If $X \notin Y$, let $Y = Y \cup \{X\}$. Otherwise, proceed with Step 5.

(4) If $X \subset E/D - E_0/D_0$, let b be the element of B such that $\mu(b) = X$, and proceed with Step 2.

(5) If the push-down stack is empty, terminate $Z = P(D[e])$. Otherwise, restore b from the push-down stack. If $\mu'(b) \in B$, let $b = \mu'(b)$ and proceed with Step 2. Otherwise, repeat this step. ∎

In order to illustrate this algorithm, its application to the structure depicted in Figures 2.1, 2.3, and 2.10 is shown below. In this example, Algorithm 3.5 is applied to determine $P(\{e_5\})$. The same values of μ are assumed for this example as were assumed for the example which was given to illustrate Algorithm 3.1. Each step is represented below by its number and the items which are affected by performing it.

(1) $X = \{\lambda\}$

$Y = \phi$

$Z = \phi$

$b = \rho(\lambda, e_5)$

(2) Stack contains $\rho(\lambda, e_5)$

$X = \{e_5\}$

$Z = \{\rho(\lambda, e_5)\}$

(5) $b = \rho(\lambda, e_5)$; Stack empty

$b = \rho(\lambda, e_4)$

(2) Stack contains $\rho(\lambda, e_4)$

$X = \{e_4, e_6\}$

(3) $Y = \{\{e_4, e_6\}\}$

(4) $b = \rho(e_4, e_1)$

(2) Stack contains $\rho(\lambda, e_4), \rho(e_4, e_1)$

$X = \{e_1, e_2, e_7, e_8\}$

(3) $Y = \{\{e_4, e_6\}, \{e_1, e_2 \cdot e_7 \cdot e_8\}\}$

(5) $b = \rho(e_4, e_1)$; Stack contains $\rho(\lambda, e_4)$

$b = \rho(e_4, e_2)$

(2) Stack contains $\rho(\lambda, e_4), \rho(e_4, e_3)$

$X = \{e_3, e_9\}$

(3) $Y = \{\{e_4, e_6\}, \{e_1, e_2, e_7, e_8\}, \{e_3, e_9\}\}$

(5) $b = \rho(e_4, e_3)$; Stack contains $\rho(\lambda, e_4)$

(5) $b = \rho(\lambda, e_4)$; Stack empty

$b = \rho(\lambda, e_6)$

(2) Stack contains $\rho(\lambda, e_6)$

$X = \{e_4, e_6\}$

(5) $b = \rho(\lambda, e_6)$; Stack empty

Note from this example that each element of B is examined exactly once. By contrast, the elements $\rho(e_4, e_1) = \rho(e_6, e_7), \rho(e_4, e_2) = \rho(e_6, e_8)$, and $\rho(e_4, e_3) = \rho(e_6, e_9)$ are each examined twice when Algorithm 3.1 is applied to the same structure. Redundant examination of these elements is avoided in Algorithm 3.5 by not examining any elements of B whose first component is $\{e_4, e_6\}$ the second time that $\{e_4, e_6\}$ is encountered (i.e., as the second component of $\rho(\lambda, e_6)$). Although $e_5 \in E_0$ in this example, no more elements of B would be examined if e_5 were not an element of E_0, because Step 3 is not performed whenever $X = \{e_5\}$.

As was noted for Algorithm 3.2, the subset of $B \cup E/D$ which is examined by Algorithm 3.5 is difficult to describe for the general application of the topological structure. Consequently, a general expression for the time which is required to perform this algorithm will not be derived. However, a method for determining this time for individual applications is illustrated in Chapter 5, where this time must be determined in order to apply the cost function to specific examples.

3.3.2.2.2 Identifying values of μ'

Since values of μ' are needed in Step 4 of Algorithm 3.1, references to these values are assumed to be stored in the blocks which represent elements of B. Then, the picture generator does not need to search the structure to compute these values. Furthermore, a

reference to each element b ϵ B such that $\mu(b) \epsilon$ E/D is assumed to be stored in the block which represents $\mu(b)$, for this reference is needed in Step 2 of Algorithm 3.1. The time which is required to perform operation ℓ_6 or ℓ_{10} is then the time which is required to examine one of these references. Generally, if the picture generator is a program, these references are pointers in a list structure, and

$$\bar{r}_{\ell}(\ell_6)\Big|_{u_1=0} = \bar{r}_{\ell}(\ell_{10})\Big|_{u_1=0} = T_p t_f/(t_f-g). \tag{3.42}$$

(Recall that T_p is the time which is required to obtain a pointer from the structure.) However, if the display processor itself is the picture generator, these references are commonly stored as display processor jump commands. An amount of time T_j' is then required to identify the block which is referenced by each jump command, and

$$\bar{r}_{\ell}(\ell_6)\Big|_{u_1=1} = \bar{r}_{\ell}(\ell_{10})\Big|_{u_1=1} = T_j' t_f/(t_f-g). \tag{3.43}$$

3.3.2.2.3 Identifying the Last Element of B Needed to Form a Subpicture

Although a reference to each element b ϵ B such that $\mu(b) \epsilon$ E/D must be stored in the block which represents $\mu(b)$, a reference to b' ϵ B such that $\mu'(b') \epsilon$ E/D is not necessarily stored in the block which represents $\mu'(b')$, since it is not needed to generate the picture. If this reference is stored (as determined by the design decision u_9 to be described in Chapter 4), it is generally implemented as a pointer in a list structure, and

$$\bar{r}_{\ell}(\ell_7)\Big|_{u_9=1} = T_p t_f/(t_f-g). \tag{3.44}$$

However, if this reference is not stored, a portion of the structure must be searched in order to perform Operation ℓ_7. The shortest search which can be used to perform this operation is described by the following algorithm:

Algorithm 3.6

Procedure for identifying $b \in B$ such that $\mu'(b) = D[e]$ for a given class $D[e] \in E/D - E_0/D_0$.

(1) Let b be the element of B such that $\mu(b) = D[e]$. (Operation ℓ_6.)

(2) Let $b = \mu'(b)$. (Operation ℓ_{10}). If $\mu'(b) = D[e]$, terminate. Otherwise, repeat this step. ∎

The first step of this algorithm is performed only once. If it is assumed that Operation ℓ_7 is applied to classes in $E/D - E_0/D_0$ with equal probability, the average number of times that Step 2 is performed is $|B|/(|E/D| - |E_0/D_0|)$. The average time which is required to perform Operation ℓ_7 by Algorithm 3.6 is then

$$\bar{r}_\ell(\ell_7)\Big|_{u_9=0} = \bar{r}_\ell(\ell_6) + (\bar{r}_\ell(\ell_{10}) + T_c t_f/(t_f - g))\,|B|/(|E/D| - |E_0/D_0|). \quad (3.4$$

3.3.2.2.4 Retrieving $G(D[e])^{-1}$ From $D[e] \in E_0/D_0$

As has been mentioned above in Section 3.1.3, each matrix $G(D[e])^{-1}$ is stored explicitly only for $D[e] \in E_0/D_0$. Consequently, operation ℓ_8 is defined only for classes in E_0/D_0. Since the matrix $G(D[e])^{-1}$ must be stored in each block which represents $D[e] \in E_0/D_0$

in any implementation of the structure $\{B, E/D, \mu, \gamma\}$, $\bar{r}_{\ell}(\ell_8)$ is generally proportional to $\bar{s}_g$, the average storage which this matrix occupies, and is independent of the references to elements of $B \cup E/D$ which are stored in the structure. The value of $\bar{r}_{\ell}(\ell_8)$ depends on the format which $G(D[e])^{-1}$ is stored for $D[e] \in E_0/D_0$, which, in turn, depends on whether the computer or the display processor performs Algorithm 3.1. An expression for $\bar{r}_{\ell}(\ell_8)$ will be given in Chapter 4.

3.3.2.2.5 Identifying Values of μ

The time which is required to perform operation ℓ_9 depends on whether or not references to values of μ are stored in the blocks which represent elements of B. Since values of μ are not needed to generate the picture, references to values of μ are generally stored as pointers, if they are stored at all. Consequently, if these references are stored (as determined by the design decision u_8 to be described in Chapter 4),

$$\bar{r}_{\ell}(\ell_9)\Big|_{u_8=1} = T_p t_f/(t_f-g). \tag{3.46}$$

However, if these references are not stored, values of μ must be determined by searching the structure. An algorithm for performing ℓ_9 in this way is the following:

Algorithm 3.7

Procedure for identifying $\mu(\rho(e,e'))$ for a given element $\rho(e,e') \in B$.

(1) Let $d = D[e]$, the first component of $\rho(e,e')$ (Operation ℓ_{11}).

(2) Let b be the element of B such that $\mu(b) = d$ (Operation ℓ_6).

If $b = \mu(e,e')$, terminate with $d = \mu(e,e')$).

(3) If $\mu'(b)=\rho(e,e')$, terminate with $b=\mu(\rho(e,e'))$. Otherwise, let $b=\mu'(b)$ (Operation ℓ_{10}) and repeat this step. ∎

Steps 1 and 2 of this algorithm are each performed only once. If Operation ℓ_9 is assumed to be applied to elements of B with equal probability, the average number of times that Step 3 is performed is $(|B|/(|E/D| - |E_0/D_0|) - 1)/2$. (Note that Step 3 is never performed with b such that $\mu'(b) = D[e]$.) Then, the average time which is required to perform Operation ℓ_9 by Algorithm 3.7 is

$$\bar{r}_\ell(\ell_9)\Big|_{u_8=0} = \bar{r}_\ell(\ell_{11}) + \bar{r}_\ell(\ell_6) + T_c t_f/(t_f-g) + (\bar{r}_\ell(\ell_{10}) + T_c t_f/(t_f-g))(|B|/(|E/D| - |E_0/D_0|) - 1)/2. \quad (3.47)$$

Since $\bar{r}_\ell(\ell_6) = \bar{r}_\ell(\ell_{10})$,

$$\bar{r}_\ell(\ell_9)\Big|_{u_8=0} = \bar{r}_\ell(\ell_{11}) + (\bar{r}_\ell(\ell_{10}) + T_c t_f/(t_f-g))(1 + |B|/(|E/D| - |E_0/D_0|))/2. \quad (3.48)$$

3.3.2.2.6 Identifying the First Component of an Element of B

The time which is required to perform Operation ℓ_{11} depends on whether or not a reference to D[e] is stored in each block which represents an element $\rho(e,e') \in B$ (as determined by the design decision u_6, which is described in Chapter 4). If this reference is stored, it is generally stored in the form of a pointer, and

$$\bar{r}_\ell(\ell_{11})\Big|_{u_6=1} = T_p t_f/(t_f-g). \quad (3.49)$$

If this reference is not stored, Operation ℓ_{11} may be performed by the following algorithm:

Algorithm 3.8

Procedure for identifying D[e] for a given element $\rho(e,e') \in B$.

(1) Let $b = \rho(e,e')$.

(2) Let $b = \mu'(b)$. If $b \in E/D$, terminate with $b = D[e]$.

Otherwise, repeat this step.

If it is assumed that Operation ℓ_{11} is applied to various elements of B with equal probability, the average number of times that Step 2 of this algorithm is performed is $(1+ |B|/(|E/D|-|E_0/D_0|))/2$. Then, if the time which is required to perform Step 1 is assumed to be insignificant, the average time which is required to perform Operation ℓ_{11} by this algorithm is

$$\bar{r}_{\ell}(\ell_{11})\Big|_{u_6=0} = (\bar{r}_{\ell}(\ell_{10})+T_b t_f(t_f-g))(1+|B|/(|E/D|-|E_0/D_0|))/2. \tag{3.50}$$

T_b denotes the time which is required to determine whether a block represents an element of B, E_0/D_0, or $E/D-E_0/D_0$.

3.3.2.2.7 Identifying the Second Component of an Element of B

A reference to the second component $D[e']$ of each element $\rho(e,e') \in B$ is commonly stored in the block which represents that element. (Whether or not this reference is stored is determined by the design decision u_7, which is described in Chapter 4.) If the picture generator is a program, this reference is generally stored as a pointer, and

$$\bar{r}_{\ell}(\ell_{12})\Big|_{\substack{u_7=1\\u_1=0}} = T_p t_f/(t_f-g). \tag{3.51}$$

However, if the display processor itself generates the picture, this reference may be stored in a different format. This format is not necessarily the same as that used to store references to values of μ'. In particular, references to values of μ' are commonly formatted as display processor jump commands, whereas references to second

components of elements of B are formatted as display processor subroutine calls. The reason for this difference is that the return pointers which are stored when the subroutine calls are executed form the push-down stack which is needed in Algorithm 3.1. If T_s' denotes the time which is required to identify the block which a display processor subroutine call references,

$$\bar{r}_{\ell}(\ell_{12})\Big|_{\substack{u_7=1\\u_1=1}} = T_s' t_f/(t_f-g). \tag{3.52}$$

However, if each set P(D[e]) is represented by a ring of storage blocks, a reference to D[e'] may not be stored in the block which represents $\rho(e,e')$. In the absence of this reference, the second component D[e'] of an element $\rho(e,e') \in B$ is identified as the element of E/D which is part of the ring which contains the block which represents $\rho(e,e')$. An algorithm for performing Operation ℓ_{12} in this way is the following:

Algorithm 3.9

Procedure for identifying the second component D[e'] of $\rho(e,e')$ from the ring which represents P(D[e']).

(1) Let $b = \rho(e,e')$.

(2) Let b be the element of $B \cup E/D$ which is referenced by the forward pointer which is stored in the block which represents b. If $b \in E/D$, terminate with $b = D[e']$. Otherwise, repeat this step. ∎

The average number of times that Step 2 of this algorithm is performed is $(1+ |B|/|E/D|)/2$. if the time which is required to perform Step 1 is assumed to be insignificant, the average time which is required to perform Operation ℓ_{12} by Algorithm 3.9 is

$$\bar{r}_{\ell}(\ell_{12})\Big|_{u_7=0} = (T_p+T_b)(t_f/(t_f-g))(1+|B|/|E/D|)/2. \quad (3.53)$$

3.3.2.2.8 Retrieving Values of F' and γ

Operations ℓ_{13} and ℓ_{14} each involve examining a block which represents an element of B to obtain information which is stored within that block. Consequently, $\bar{r}_{\ell}(\ell_{13})$ and $\bar{r}_{\ell}(\ell_{14})$ do not depend on the references to elements of $B \cup E/D$ which are stored in the structure. Generally, the average time $\bar{r}_{\ell}(\ell_{13})$ which is required to determine $F'(\rho(e,e'))$ is proportional to $\bar{s}_f$. The time which is required to perform Operation ℓ_{14}, however, may vary radically with the way in which the structure is implemented. For example, $\bar{r}_{\ell}(\ell_{14})$ is the time which is required to read the high order half of the third word in the list structure shown in Figure 3.2. However, in the program which is described in Appendix A, the application of Operation ℓ_{14} to $\rho(e,e') \in B$ involves determining the length of the storage block which represents $\rho(e,e')$ if either $\gamma(\rho(e,e')) = (I, G(D[e']), I)$ or $\gamma(\rho(e,e')) = (G(D[e'])^{-1}, G(D[e']), G(D[e'])^{-1})$. Otherwise, the length of the storage block must first be determined to establish that either $\gamma(\rho(e,e')) = (G(D[e'])^{-1}, G(D[e']), I)$ or

$\gamma(\rho(e,e')) = (I, G(D[e']), G(D[e'])^{-1})$, and then the two push jump commands in the block must be compared to select one of these two values. Like $\bar{r}_\ell(\ell_8)$, the average time which is required to perform each of these basic operations depends on whether the computer or the display processor performs Algorithm 3.1. Expressions for $\bar{r}_\ell(\ell_{13})$ and $\bar{r}_\ell(\ell_{14})$ will be given in Chapter 4.

3.4 Conclusion

The cost of an implementation of the topological structure was defined to be the average time which is required to operate on the structure in order to respond to a control language input. The constraints which restrict the set of possible implementations of the structure were defined to be limitations on both the amount of storage which is available to contain the structure and the amount of time which elapses while the picture is generated once. The storage required by an implementation of the structure, the interference of the picture generation process with the response to control language inputs, and the cost itself were then discussed. Before this discussion can be applied to compare various implementations of the topological structures, however, the cost function and constraints must be described as functions of the various design decisions. The cost function and constraints are expressed in this way in the following chapter.

Chapter 4

DESIGN DECISIONS

The discussion in Chapter 3 may be applied to determine the optimum implementation of the topological structure for a given equipment configuration and application. As a preliminary step, however, the cost must be described as a function of various design decisions. In order to describe the cost in this way, each design decision is considered to be a binary - valued variable. The value of each design decision represents a choice of one of two possible methods of implementing some feature of the topological structure. The optimum implementation of the topological structure is then described by the set of values for these design decisions which minimizes the cost.

The design decisions which will be considered are defined below. The storage s which is required by the topological structure, the total time g which is arrested from the computer in order to generate the picture, the time t_f which elapses while the picture is generated, and the cost $\bar{r}$ are then expressed as functions of these design decisions. Finally, the constraints which apply to the design decisions are described.

4.1 Definition of a Set of Design Decisions

A list of suitable design decisions for the topological structure is shown in Table 4.1. Each of these decisions may

Design Decision	Interpretation
u_1	The display processor, rather than the computer, performs Algorithm 3.1.
u_2	The display processor, rather than the computer, transforms subpictures to produce entities in E_0.
u_3	The topological structure is $\{B, E/D, \mu, \gamma\}$, rather than $\{B, E/D, F'\}$.
u_4	Operation ℓ_5 is performed by Algorithm 3.5.
u_5	Values of H are saved on the push-down stack in Algorithm 3.1.
u_6	A reference to the block which represents D[e] is stored in the block which represents $\rho(e,e')$ for all $\rho(e,e') \epsilon B$.
u_7	A direct reference to the block which represents D[e'] is stored in the block which represents $\rho(e,e')$ for all $\rho(e,e') \epsilon B$.

Table 4.1

Design Decisions

Design Decision	Interpretation
u_8	A reference to the block which represents $\mu(b)$ is stored in the block which represents b for all $b \in B$.
u_9	A reference to the block which represents $b \in B$ such that $\mu'(b) = D[e]$ is stored in the block which represents $D[e]$ for all $D[e] \in E/D - E_0/D_0$.
u_{10}	The blocks which represent the elements of each set $P(D[e])$ are linked together in a ring with the block which represents $D[e]$.
u_{11}	Both forward and backward pointers are stored in each block in the ring which links the blocks which represent elements of each set $P(D[e])$ with the block which represents $D[e]$.

Table 4.1 (cont.).

Design Decisions

assume only the value 0 or 1. The interpretation which is given in the table for each decision is the interpretation for that decision when it assumes the value 1. The opposite interpretation is assumed if the value of the decision is 0. For example, $u_3 = 1$ implies that the structure is $\{B, E/D, \mu, \gamma\}$, whereas $u_3 = 0$ implies that the structure is $\{B, E/D, F'\}$.

The design decisions u_1 and u_2 describe how the picture generator is implemented. If $u_1 = u_2 = 0$, the picture generator is entirely a program. Because of hardware restrictions, this type of picture generator was used in early computer graphics programs, e.g., SKETCHPAD [67,68] and SKETCHPAD III [28]. Although most later display processors did not impose as severe restrictions, some more recent computer graphics programs, e.g., PENCIL [74], PLAN-PGS [11], and DIM [5], have also used this type of picture generator. If $u_1 = 0$ and $u_2 = 1$, the picture generator is partly a program and partly display processor hardware. The picture generator for GRAPHIC -2 [12] was implemented in this manner. If $u_1 = u_2 = 1$, the picture generator is entirely display processor hardware. This type of picture generator has been used by the author in the program which is described in Appendix A, as well as in earlier efforts [26]

Although no attempt to implement a picture generator for which $u_1 = 1$ and $u_2 = 0$ is known to the author, such a picture generator is certainly conceivable. For example, consider a three-dimensional display processor which is capable of drawing vectors which are specified by x,y, and z components, but which does not include a rotation capability. Furthermore, assume that a topological structure which involves only coordinate transformations which are translations in the x,y, and z directions may be implemented with $u_1 = u_2 = 1$. However, suppose that this display processor is to be used to display two-dimensional pictures which involve only translations in the x and y directions and rotations in the xy plane. Then, the topological structure may be implemented in the same way, with the exception that entities in E_0 are formed by the program (i.e., $u_2 = 0$) so that the z coordinate is interpreted as a rotation angle, rather than as a third coordinate.

The value of u_3 is not obvious for many implementations. As has been mentioned in Chapter 2, artificial entities must be introduced into the structure $\{B, E/D, \mu, \gamma\}$ to permit some entities to be modified without transforming the coordinates of other entities. Similarly, artificial entities must be introduced into the structure $\{B, E/D, F'\}$ to provide for the enforcement of some concatenation constraints as a consequence of generating the picture. Many applications require both that some concatenation constraints be enforced and that some entities be treated independently of other

entities. Consequently, which form the topological structure should assume often cannot be determined by inspection.

Because the artifical entities which must be represented by the structure are determined by the value of u_3, the cardinalities of the sets B, E/D, and E_0/D_0, as well as the sums $\sum_{e\epsilon E-\{\lambda\}} \psi(e)$ and $\sum_{e\epsilon E_0} \psi(e)$, depend on the value of u_3. This fact may be expressed as follows:

$$|B| = (1-u_3)\, q_{bf} + u_3\, q_{bg}, \tag{4.1}$$

$$|E/D| = (1-u_3)\, q_{df} + u_3\, q_{dg}, \tag{4.2}$$

$$|E_0/D_0| = (1-u_3)\, q_{d0f} + u_3\, q_{d0g} \tag{4.3}$$

$$\sum_{e\epsilon E-\{\lambda\}} \psi(e) = (1-u_3)\, q_{sf} + u_3 q_{sg}, \tag{4.4}$$

and

$$\sum_{e\epsilon E_0} \psi(e) = (1-u_3)\, q_{s0f} + u_3\, q_{s0g}, \tag{4.5}$$

where

q_{bf} is the cardinality of B for the structure $\{B, E/D, F'\}$,

q_{bg} is the cardinality of B for the structure $\{B, E/D, \mu, \gamma\}$,

q_{df} is the cardinality of E/D for the structure $\{B, E/D, F'\}$,

q_{dg} is the cardinality of E/D for the structure $\{B, E/D, \mu, \gamma\}$,

q_{d0f} is the cardinality of E_0/D_0 for the structure $\{B, E/D, F'\}$,

q_{d0g} is the cardinality of E_0/D_0 for the structure $\{B, E/D, \mu, \gamma\}$,

q_{sf} is $\sum_{e \in E-\{\lambda\}} \psi(e)$ for the structure $\{B, E/D, F'\}$,

q_{sg} is $\sum_{e \in E-\{\lambda\}} \psi(e)$ for the structure $\{B, E/D, \mu, \gamma\}$,

q_{s0f} is $\sum_{e \in E_0} \psi(e)$ for the structure $\{B, E/D, F'\}$, and

q_{s0g} is $\sum_{e \in E_0} \psi(e)$ for the structure $\{B, E/D, \mu, \gamma\}$.

The various coefficients of u_3 and $(1-u_3)$ in these equations are determined from the form which the topological structure assumes for the application under consideration, as will be demonstrated in Chapter 5.

The design decision u_4 may assume the value 1 only if $u_{10} = 0$, since the purpose of linking the blocks which represent the elements of each set P(D[e]) together in a ring with the block which represents D[e] is to allow operation ℓ_5 to be performed by a much simpler algorithm than Algorithm 3.5. If $u_4 = u_{10} = 0$, the picture generator is assumed to identify the elements of each set P(D[e]) to a program which is applying operation ℓ_5 to the class D[e] .

Since there is no advantage to saving matrices on the push-down stack in Algorithm 3.1 when $u_3 = 1$, $u_3 = 1 \Longrightarrow u_5 = 0$. If $u_3 = u_5 = 0$,

the inverse of a matrix which is a value of F' must be determined whenever Step 4 of Algorithm 3.1 is performed.

The remaining design decisions $u_6, \ldots, u_{11}$ specify which nonessential references to elements of $B \cup E/D$ are stored in the structure in order to reduce the average time which is required to perform certain basic operations. One of these parameters, u_7, affects the time which is required to generate the picture, as it affects $\bar{g}_x$, the average time during which the computer is devoted to the picture generation process in order to identify the second component of an element of B. Clearly, $u_{10} = 0 \Longrightarrow u_{11} = 0$.

4.2 Effect of Design Decisions on Storage

The average storage $\bar{s}_b$ which is required to represent an element of B, the average storage $\bar{s}_{d0}$ which is required to represent an element of E_0/D_0, and the storage s_d which is required to represent an element of $E/D-E_0/D_0$ have been described by Equations 3.4, 3.6, and 3.7 as sums of smaller storage terms. Each of these equations contains a term which represents storage which is unusable because the various items which are to be stored cannot be packed arbitrarily. Because many of the design decisions describe which items are to be stored in each type of storage block, this average unusable storage is a function of many design decisions, and, therefore, is difficult to describe. However, if certain assumptions are made about the way in which the items in each block may be packed, an algorithm may be specified to determine the size of that storage block without actually computing the

unusable storage which it contains.

4.2.1 An Algorithm for Computing Sizes of Storage Blocks

The items which are stored in a block which represents an element of $B \cup E/D$ will be assumed to be packed subject to the following restrictions:

(1) Each item which occupies more than one storage location occupies consecutive locations.

(2) Only the first location which an item occupies may be only partially occupied by that item. The purpose of this restriction is to allow the format of an item to remain fixed, regardless of the way in which it is packed with other items.

The first of these restrictions does not appreciably limit the flexibility of the implementation. If the display processor performs Algorithm 3.1, each item must be stored in consecutive locations so that the display processor can access the item as a unit. If the computer performs Algorithm 3.1, each item is more easily accessed if it is stored in consecutive locations, rather than in pieces which are scattered throughout the block. The second restriction, however, does limit the flexibility of the implementation if certain items must be stored in a given part of a storage location. For example, if each display processor command must be stored in the low order half of a location, the last n-1 locations of those which are occupied by an item which is

described by n commands must be considered to be completely occupied. Consequently, another item which is not represented by a display processor command cannot be stored in the vacant half of any of these locations.

Whenever these restrictions are satisfied, the following algorithm may be applied to determine the size of the smallest storage block which can contain n given items. In the discussion of this algorithm, $[x]$ denotes the greatest integer less than or equal to x.

Algorithm 4.1

Procedure for determining the minimum number x of storage locations which are required to hold n items whose individual storage requirements are $x_1, x_2, \ldots, x_n$ locations. $x_1, x_2, \ldots, x_n$ are not necessarily integers, but are assumed to be ordered such that $x_i - [x_i] \geq x_{i+1} - [x_{i+1}]$ for $i = 1, 2, \ldots, n-1$. (This ordering is not necessarily unique.)

(1) Let $i = 1$ and $j = n$.

(2) If $j = i$, let $x = -\sum_{k=1}^{i} [-x_k]$ and terminate.

(3) If $x_i = [x_i]$, let $i = i+1$ and proceed with Step 2.

(4) If $x_j = [x_j]$ or $x_i + x_j \geq [x_i] + 1$ or $x_i + x_j \geq [x_j] + 1$, let $x_i = x_i + x_j$ and let $j = j - 1$. Otherwise, let $i = i + 1$. Proceed with Step 2. ∎

As an example of this algorithm, consider the storage of 7 items whose individual storage requirements are the following:

$x_1 = 1.5$ locations

$x_2 = 0.4$ locations

$x_3 = 2.3$ locations

$x_4 = 0.3$ locations

$x_5 = 1.25$ locations

$x_6 = 0.25$ locations

$x_7 = 1.0$ locations

These lengths are ordered as required for Algorithm 4.1. Each step of this algorithm as it applies to these values is indicated below by its number and the items which are affected by performing it.

(1) $i = 1$

$j = 7$

(4) $x_1 = 2.5$

$j = 6$

(4) $x_1 = 2.75$

$j = 5$

(4) $i = 2$

(4) $x_2 = 1.65$

$j = 4$

(4) $x_2 = 1.95$

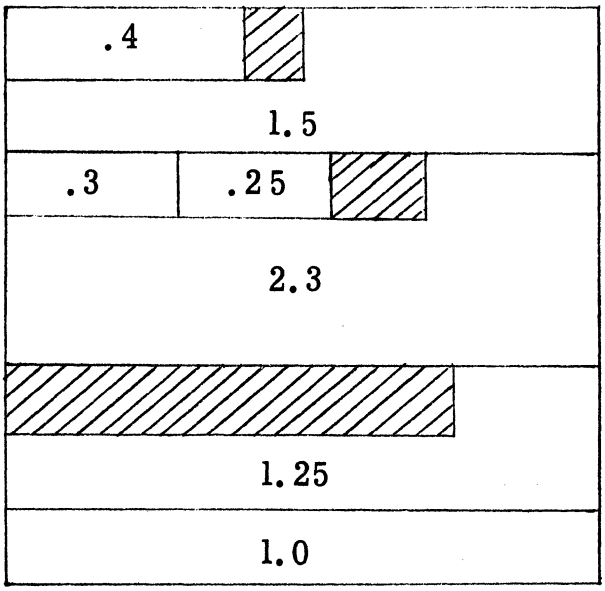
$j = 3$

(4) $i = 3$

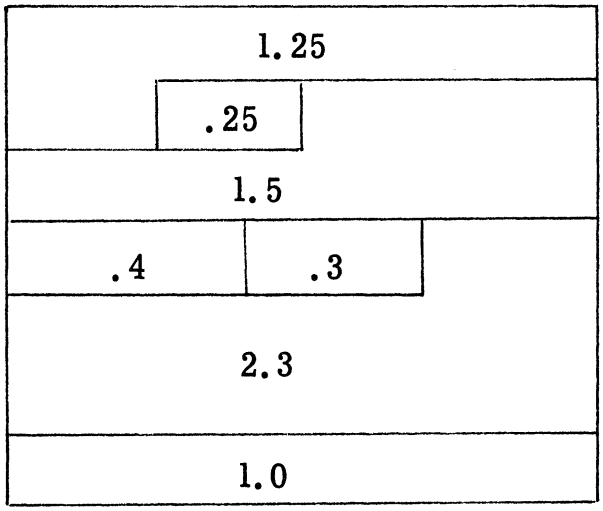
(2) $x = -\ [\ -2.75] + [\ -1.95] + [\ -2.3]) = 8$ locations

Note that 8 locations are needed to store the seven items in this example, although $\sum_{i+1}^{7} x_i = 7$. The additional location is needed to insure that all locations, other than the first one, which are occupied by a particular item are completely occupied by that item. To illustrate this point, a minimal storage arrangement is shown in Figure 4.1a, and a fully packed storage arrangement is shown in Figure 4.1b. In these figures, the storage which is occupied by each item is indicated within that item. If the items were packed as shown in Figure 4.1b, only 7 locations would be required to store the 7 items. However, in violation of the second restriction stated above, the item which occupies 1.25 locations would then occupy only part of the last location in which it is stored. Although other minimal storage arrangements than the one shown in Figure 4.1a are possible, any minimal storage arrangement for this example requires 8 locations.

Algorithm 4.1 may be applied to help compute the average storage which is required by a block which represents an element of $B, E_0/D_0$, or $E/D\text{-}E_0/D_0$. For Algorithm 4.1 to be applicable to this task, each item which is stored in each block which represents an element of one of the sets $B, E_0/D_0$, or $E/D\text{-}E_0/D_0$ is assumed to occupy either (1) the same number of locations in each block which represents an element of the set, o



a. A Minimal Storage Arrangement



b. A Fully Packed Storage Arrangement

Figure 4.1

Packing of Items Into Storage Blocks

(2) an integer number of locations in each block which represents an element of the set. With this assumption, the total storage per block which is occupied by those items which occupy the same amount of storage in each block may be computed by Algorithm 4.1. The average storage per block is then this storage plus the average amount of storage which is occupied by the items which do not occupy the same amount of storage in each block.

The average storage $\bar{s}_b$ which is required to represent an element of B has been described by Equation 3.4. The only terms other than $\bar{s}_{wb}$ in this equation which are averages are $\bar{s}_f$ and $\bar{s}_\gamma$. If these terms are assumed to be averages of integers, Algorithm 4.1 may be applied to determine the sum of the other terms. This sum may then be added to $\bar{s}_f$ and $\bar{s}_\gamma$ to yield $\bar{s}_b$. In this way, the necessity to compute $\bar{s}_{wb}$ is avoided. Generally, each value of F' does occupy an integer number of storage locations. However, each value of γ commonly occupies less than one storage location. For example, each value of γ may be represented by two bits in a list structure. If each value of γ does not occupy an integer number of locations, but it occupies the same amount of storage in each block which represents an element of B, Algorithm 4.1 may be applied to compute the sum of all terms in Equation 3.4 except $\bar{s}_f$. Then, $\bar{s}_f$ may be added to this sum to yield $\bar{s}_b$.

The average storage $\bar{s}_{d0}$ which is required to represent an element of E_0/D_0 has been described by Equation 3.6. The terms other than

$\bar{s}_{wd}$ in this equation which represent averages are $\bar{s}_{\beta 0}$ and $\bar{s}_{g}$. Generally, the number of locations which each subpicture which is associated with a class in E_0/D_0 occupies is an integer. Furthermore, each matrix $G(D[e])^{-1}$ for $D[e] \in E_0/D_0$ is generally stored in an integer number of locations. Consequently, $\bar{s}_{wd0}$ is independent of the storage which is occupied by these items, and Algorithm 4.1 may be applied to the terms s_0, s_t, s_p, and s_{r0} to yield the sum $s_0 + s_t + s_p + s_{r0} + \bar{s}_{wd0}$. Then, this sum may be added to $\bar{s}_{\beta 0}$ and $\bar{s}_{g}$ to yield $\bar{s}_{d0}$.

The storage s_d which is required to represent an element of $E/D - E_0/D_0$ has been described by Equation 3.7. Since s_d represents the actual storage which is occupied by a block, rather than an average amount of storage, it may be computed directly by applying Algorithm 4.1 to s_0, s_t, s_p, s_r, and s_β. Thus, the necessity to compute s_{wd} is avoided.

The above discussion has indicated methods for computing $\bar{s}_b$, $\bar{s}_{d0}$, and s_d without first determining $\bar{s}_{wb}$, $\bar{s}_{wd0}$, or s_{wd}. In this discussion, each term in Equations 3.4, 3.6, and 3.7 has been assumed to be the storage which is occupied by one item. However, if $u_{10} = u_{11} = 1$, s_p is the storage which is occupied by both forward and backward pointers to elements of a ring which represents a set $P(D[e])$, and, if $u_9 = 1$, s_β is the storage which is occupied by references to both the first and last elements of a set $\rho(\{e\} \times S(e))$.

Then,

$$s_p = s_p' + s_p'', \tag{4.6}$$

where

s_p' is the storage which is occupied in each block which represents an element of $B \cup E/D$ by a forward pointer to an element of a ring which represents a set $P(D[e])$, and

s_p'' is the storage which is occupied in each block by a backward pointer to an element of a ring which represents a set $P(D[e])$.

Furthermore,

$$s_\beta = s_\beta' + s_\beta'', \tag{4.7}$$

where

s_β' is the storage which is occupied in each block which represents an element $D[e] \in E/D - E_0/D_0$ by a reference to the first element of the set $\rho(\{e\} \times S(e))$, and

s_β'' is the storage which is occupied in each block which represents an element $D[e] \in E/D - E_0/D_0$ by a reference to the last element of the set $\rho(\{e\} \times S(e))$.

In order to provide for the most efficient packing of this information, s_p' and s_p'', rather than s_p, will be considered to be arguments to Algorithm 4.1. Similarly, s_β' and s_β'', rather than s_β, will be considered to be arguments to Algorithm 4.1.

The storage quantities needed to compute $\bar{s}_b$ are then s_0, s_t, s_p', s_p'', s_e, s_e', s_μ, s_μ', $\bar{s}_f$, and $\bar{s}_\gamma$, those needed to compute $\bar{s}_{d0}$ are s_0, s_t, s_p', s_p'', s_{r0}, $\bar{s}_{\beta 0}$, and $\bar{s}_g$, and those needed to compute s_d are s_0, s_t, s_p', s_p'', s_r, s_β', and s_β''. These storage quantities are described below as functions of the design parameters.

4.2.2 Storage of Items Common to All Blocks

The overhead storage s_0 which is associated with creating and destroying each storage block may assume several different forms. For example, some storage may be reserved at each end of each storage block as shown in Figure 3.1. Alternatively, overhead storage may be reserved at only one end of each block. This storage could contain such information as the length of the block and a flag to indicate whether or not the block is in use. The overhead storage which is associated with a particular block may even appear in a table which is external to the block itself. Because the overhead storage may assume many forms, the way in which other information may be packed with the overhead information is difficult to parameterize. Consequently, this overhead storage will be considered to contain only items which occupy integer numbers of locations.

The storage s_t which is needed in each block which represents an element of $B \cup E/D$ to hold information which describes whether the block represents an element of $B, E/D - E_0/D_0$, or E_0/D_0 depends on whether or not the display processor performs Algorithm 3.1. As has been mentioned in Section 3.1, s_t may be zero if the display processor performs this algorithm and the type of element which each block represents may be determined from its format. Similarly, s_t may be zero if the computer performs Algorithm 3.1 and the type of element which each block represents may be determined from some feature of the block, as in the examples described in Section 3.1.4. Consequently,

$$s_t = (1 - u_1)\, S_t + u_1 S_t', \tag{4.8}$$

where

S_t is the storage in each block which is occupied by information which identifies the type of element that each block represents if the computer performs Algorithm 3.1, and

S_t' is the storage which is occupied by this information if the display processor performs Algorithm 3.1.

Generally, if $S_t \neq 0$ and $S_t' \neq 0$, $S_t = S_t' = 2$ bits.

The storage quantities s_p' and s_p'' which are required to link each block in a ring which represents a set P(D[e]) to adjacent blocks in the ring are zero if $u_{10} = 0$. If $u_{10} = 1$, s_p' is the storage which is

occupied by a pointer. If $u_{10} = u_{11} = 1$, s_p'' is also the storage which is occupied by a pointer. However, if $u_{10} = 1$ and $u_{11} = 0$, $s_p'' = 0$. Then, if S_p denotes the storage which is occupied by a pointer,

$$s_p' = u_{10} S_p, \tag{4.9}$$

and

$$s_p'' = u_{11} s_p'. \tag{4.10}$$

Recall that $u_{11} = 1 \Longrightarrow u_{10} = 1$.

4.2.3 Storage of Items Which Describe an Element of B

The storage s_e which is required to reference the first component of an element of B is S_p if these references are stored, or zero otherwise. Consequently,

$$s_e = u_6 S_p. \tag{4.11}$$

Similarly, the storage s_μ which is required to reference a value of μ is S_p if references to values of μ are stored, or zero otherwise, i.e.,

$$s_\mu = u_8 S_p. \tag{4.12}$$

However, the storage s_e' which is required to store a reference to the second component of an element of B and the storage s_μ' which is required to store a reference to a value of μ' depend on whether the display processor or the computer performs Algorithm 3.1.

If the display processor performs Algorithm 3.1, s_e' is the storage which is required to hold a display processor subroutine call command,

whereas s_μ' is the storage which is required to hold a display processor jump command. These quantities will be denoted S_s and S_j, respectively. (If the display processor command set does not include one of these commands, u_1 is constrained to be zero.) If the computer performs Algorithm 3.1 and $u_{10} = 1$, u_7 may be zero. Consequently, s_e' may be zero, since $D[e']$ may be identified from $\rho(e, e')$ by sequencing through the ring which represents $P(D[e'])$. The quantities s_e' and s_μ' may then be expressed as follows:

$$s_e' = u_1 S_s + u_7 (1 - u_1) S_p, \tag{4.13}$$

and

$$s_\mu' = u_1 S_j + (1 - u_1) S_p. \tag{4.14}$$

The average storage $\bar{s}_f$ which is required to represent a value of F' and the average storage $\bar{s}_\gamma$ which is required to represent a value of γ each depend on whether the computer or the display processor performs Algorithm 3.1. Furthermore, $\bar{s}_f = 0$ if the structure is $\{B, E/D, \mu, \gamma\}$, whereas $\bar{s}_\gamma = 0$ if the structure is $\{B, E/D, F'\}$. These two quantities may then be expressed as follows:

$$\bar{s}_f = (1 - u_3)(1 - u_1) S_f + (1 - u_3) u_1 S_f' \tag{4.15}$$

where

S_f is the average storage which is required to represent a value of F' if the computer performs Algorithm 3.1, and

S_f' is the average storage which is required to represent a value of

F' if the display processor performs Algorithm 3.1,

and

$$\bar{s}_{\gamma} = u_3 (1 - u_1) S_{\gamma} + u_3 u_1 S_{\gamma}', \tag{4.16}$$

where

S_{γ} is the average storage which is required to represent a value of γ if the computer performs Algorithm 3.1, and

S_{γ}' is the average storage which is required to represent a value of γ if the display processor performs Algorithm 3.1.

(S_f, S_f', S_{γ}, and S_{γ}' are estimated from both hardware specifications and the forms which the structures $\{B, E/D, F'\}$ and $\{B, E/D, \mu, \gamma\}$ assume for the application under consideration.)

4.2.4 Storage of Items Which Describe an Element of E/D

As described in Chapter 3, the subpicture which is associated with each class $D[e] \in E/D - E_0/D_0$ is represented by references to elements of B whose first component is $D[e]$. Since these elements are assigned an order for purposes of generating the picture, references to only the first and last of them are generally stored. Since the reference to the first of these elements is needed in order to generate the picture, it may be implemented as either a pointer or a display processor jump command, depending on the value of u_1. The reference to the last of these elements is optional. If it is stored, it is stored as a pointer. The amounts of storage s_{β}' and s_{β}'' which are occupied by these two references are then the following:

$$s_{\beta}' = (1 - u_1) S_p + u_1 S_j, \tag{4.17}$$

and

$$s_{\beta}'' = u_9 S_p. \tag{4.18}$$

However, for each class in E_0/D_0, the associated subpicture must be represented explicitly, either as a table of plotting information or as a display processor program. Furthermore, since a different number of artificial entities may be needed for the structure $\{B, E/D, F'\}$ than for the structure $\{B, E/D, \mu, \gamma\}$, the average storage which is occupied by one of these subpictures depends on which structure is used. Then,

$$\bar{s}_{\beta 0} = (1 - u_3)((1 - u_2) S_{\beta f} + u_2 S_{\beta f}') + u_3 ((1 - u_2) S_{\beta g} + u_2 S_{\beta g}'), \tag{4.19}$$

where

$S_{\beta f}$ is the average storage which is required to represent a subpicture which is associated with a class in E_0/D_0 as a table of plotting information for the structure $\{B, E/D, F'\}$,

$S_{\beta f}'$ is the average storage which is required to represent a subpicture which is associated with a class in E_0/D_0 as a display processor program for the structure $\{B, E/D, F'\}$,

$S_{\beta g}$ is the average storage which is required to represent a subpicture which is associated with a class in E_0/D_0 as a table of plotting information for the structure $\{B, E/D, \mu, \gamma\}$, and

$S_{\beta g}'$ is the average storage which is required to represent a subpicture which is associated with a class in E_0/D_0 as a display processor program for the structure $\{B, E/D, \mu, \gamma\}$.

The storage which is occupied by each matrix $G(D[\,e]\,)^{-1}$ for $D[\,e] \in E_0/D_0$ depends on whether the display processor or the computer performs Algorithm 3.1. Furthermore, these matrices are not stored if the structure is $\{B, E/D, F'\}$, rather than $\{B, E/D, \mu, \gamma\}$. The average storage which is occupied by one of these matrices is then

$$\bar{s}_g = u_3\,((1 - u_1)\,S_g + u_1\,S_g'), \tag{4.20}$$

where

S_g is the average storage which is occupied by a matrix $G(D[\,e]\,)^{-1}$ for $D[\,e] \in E_0/D_0$ if the computer performs Algorithm 3.1, and

S_g' is the average storage which is occupied by one of these matrices if the display processor performs Algorithm 3.1.

(S_g and S_g' are estimated from the hardware specifications and the form which the structure $\{B, E/D, \mu, \gamma\}$ assumes for the application under consideration.)

Although each matrix $G(D[\,e]\,)^{-1}$ is stored only for $D[\,e] \in E_0/D_0$, some storage may be required in each block which represents a class $D[\,e] \in E/D - E_0/D_0$ to hold information which describes the computation of $G(D[\,e]\,)^{-1}$ if the display processor performs Algorithm 3.1. For example, in the program which is described in Appendix A, an extra jump command is needed for this purpose in each block which represents an element of $E/D - E_0/D_0$. The storage which is needed for this purpose will be denoted by S_a. If $u_2 = 0$ and $u_1 = 1$, a display processor command must be included in each block which represents an element of E_0/D_0 to stop the display processor and interrupt the

computer. This storage will be denoted S_h. Finally, some storage is required in each block which represents an element of E/D to hold the display processor commands which return from a display processor subroutine call. This storage will be denoted S_r. Then, the storage which is required to store display processor commands which serve only to control the display processor may be summarized as follows:

$$s_{r0} = u_1 (u_2 S_h + (1 - u_2) S_r), \tag{4.21}$$

and

$$s_r = u_1 (S_r + u_3 S_a). \tag{4.22}$$

4.3 Effect of Design Decisions on Picture Generation Time

As has been described in Section 3.3.2, the total time g during which the computer is devoted to the picture generation process while the picture is generated once and the time t_f which elapses while the picture is generated once affect the time which is required to perform basic operations. These quantities are expressed as functions of the design parameters below.

4.3.1 Time During Which the Computer is Devoted to Picture Generation

Recall from Equation 3.26 that

$$g = (\bar{g}_s + \bar{g}_r + \bar{g}_m + \bar{g}_h + \bar{g}_b + \bar{g}_n + \bar{g}_x) \sum_{e \in E - \{\lambda\}} \psi(e)$$

$$+ (\bar{g}_c + \bar{g}_t) \sum_{e \in E_0} \psi(e).$$

Each of the quantities $\bar{g}_s$, $\bar{g}_r$, $\bar{g}_m$, $\bar{g}_h$, $\bar{g}_b$, $\bar{g}_n$, $\bar{g}_x$, $\bar{g}_c$, and $\bar{g}_t$ which appears in this equation depends on one or more design parameters.

The average times $\bar{g}_s$ and $\bar{g}_r$ during which the computer must be devoted to the picture generation process in order to save items on the push-down stack and to restore these items from the push-down stack during Steps 2 and 4 of Algorithm 3.1 depend on whether the computer or the display processor performs Algorithm 3.1. Furthermore, these quantities are proportional to each other, since all items which are saved in Step 2 of this algorithm are restored in Step 4 of the algorithm. As has been mentioned in Section 3.2.1, the program may save either a reference to an element of B or both a reference to an element of B and a coordinate transformation matrix during Step 2. Consequently, $\bar{g}_s$ and $\bar{g}_r$ also depend on the design parameter u_5. Since $\bar{g}_s$ and $\bar{g}_r$ are proportional to each other, the sum $\bar{g}_s + \bar{g}_r$ which appears in Equation 3.26 may be expressed as follows:

$$\bar{g}_s + \bar{g}_r = (1 - u_1)\, G_s + u_1\, G_s' + u_5\, (1 - u_1)\, G_s'' + u_1 u_5\, G_s''', \qquad (4.23)$$

where

G_s is the time which is required for the computer to save and restore a reference to an element of B,

G_s' is the time during which the computer is suspended while the display processor saves and restores a reference to an element of B,

G_s'' is the average time which is required for the computer to save and restore each value of H, and

G_s''' is the average time during which the computer is suspended while the display processor saves and restores each value of H.

The average time $\bar{g}_m$ during which the computer is devoted to the picture generation process while a value of H is premultiplied by either a value of F' or a value of G' depends on whether the computer or the display processor performs the matrix multiplication. Furthermore, as described in Chapter 3, $\bar{g}_m$ also depends on the average complexity of values of F' or G'. Since this complexity may differ for these two functions, $\bar{g}_m$ depends on whether the structure is $\{B, E/D, F'\}$ or $\{B, E/D, \mu, \gamma\}$. Since second components of elements of B must be identified in order to determine values of G', $\bar{g}_m$ also depends on whether or not a reference to the second component of each element of B is stored in the block which represents that element, as described by the value of u_7. To summarize,

$$\bar{g}_m = (1 - u_3)\,((1 - u_1)\,G_{mf} + u_1\,G_{mf}') + u_3\,((1 - u_1)\,(u_7 G_{mg} + (1 - u_7)\,G_{mg}'') + u_1\,G_{mg}'), \qquad (4.24)$$

where

G_{mf} is the average time which is required for the computer to premultiply a value of H by a value of F',

G_{mf}' is the average time during which the computer is suspended

while the display processor premultiplies a value of H by a value of F',

G_{mg} is the average time which is required for the computer to premultiply a value of H by a value of G' if u_7=1,

G_{mg}' is the average time during which the computer is suspended while the display processor premultiplies a value of H by a value of G', and

G_{mg}'' is the average time which is required for the computer to premultiply a value of H by a value of G' if $u_7 = 0$.

(G_{mf}, G_{mf}', G_{mg}, G_{mg}', and G_{mg}'' are estimated from both hardware specifications and the forms which the structures $\{B, E/D, F'\}$ and $\{B, E/D, \mu, \gamma\}$ assume for the application under consideration.)

The average time $\bar{g}_h$ during which the computer is devoted to the picture generation process while a value of H is premultiplied by the inverse of a value of F' or by a value of G'' is also determined by the design parameters u_1, u_3, and u_7. However, $\bar{g}_h = 0$ for the structure $\{B, E/D, F'\}$ if premultiplication by inverses of values of F' is avoided by saving values of H on the push-down stack. Consequently,

$$\bar{g}_h = (1 - u_5)(1 - u_3)((1 - u_1) G_{hf} + u_1 G_{hf}') + u_3 ((1 - u_1)(u_7 G_{hg} + (1 - u_7) G_{hg}'') + u_1 G_{hg}'), \quad (4.25)$$

where

G_{hf} is the average time which is required for the computer to premultiply a value of H by the inverse of a value of F',

G_{hf}' is the average time during which the computer is suspended

while the display processor premultiplies a value of H by the inverse of a value of F',

G_{hg} is the average time which is required for the computer to premultiply a value of H by a value of G'' if $u_7 = 1$,

G_{hg}' is the average time during which the computer is suspended while the display processor premultiplies a value of H by a value of G'', and

G_{hg}'' is the average time which is required for the computer to premultiply a value of H by a value of G'' if $u_7 = 0$.

($G_{hf}, G_{hf}', G_{hg}, G_{hg}'$, and G_{hg}'' are estimated from hardware specifications and from the forms which the structures $\{B, E/D, F'\}$ and $\{B, E/D, \mu, \gamma\}$ assume for the application under consideration.)

The average time $\bar{g}_b$ during which the computer is devoted to the picture generation process while the element $b \in B$ such that $\mu(b) = D[e]$ for a given class $D[e]$ is identified depends on u_1, since u_1 describes whether this reference is stored as a pointer or as a display processor jump command. Furthermore, if $u_1 = 0$, the computer must determine whether or not b exists before it attempts to identify this element. The average time which is required for this process depends on the relative cardinalities of E_0/D_0 and E/D. Since these cardinalities are functions of u_3, $\bar{g}_b$ also depends on u_3. To summarize,

$$\bar{g}_b = (1 - u_1)\,((1 - u_3)\,G_{bf} + u_3\,G_{bg}) + u_1\,T_j, \qquad (4.26)$$

where

G_{bf} is the average time which is required for the computer to determine whether or not a class $D[e]$ belongs to E_0/D_0,

and, if it does not, to identify $b \in B$ such that $\mu(b) = D[e]$ for the structure $\{B, E/D, F'\}$,

G_{bg} is the average time which is required for the computer to determine whether or not a class $D[e]$ belongs to E_0/D_0, and, if it does not, to identify $b \in B$ such that $\mu(b) = D[e]$ for the structure $\{B, E/D, \mu, \gamma\}$, and

T_j is the time during which the computer is suspended while a jump command is transferred to the display processor.

(G_{bf} and G_{bg} are estimated from both hardware specifications and the forms which the structures $\{B, E/D, F'\}$ and $\{B, E/D, \mu, \gamma\}$ assume for the application under consideration.)

The only design parameter which affects the average time $\bar{g}_n$ during which the computer is devoted to the picture generation process while $\mu'(b)$ is being identified for a given $b \in B$ is u_1, i.e.,

$$\bar{g}_n = (1 - u_1)\ (T_p + T_b) + u_1\, T_j. \tag{4.27}$$

(Recall that T_p is the time which is required to read a pointer from the structure and that T_b is the time which is required to determine whether a storage block represents an element of B, E_0/D_0, or $E/D - E_0/D_0$.) If $u_1 = 0$, the computer reads the pointer to the block which represents $\mu'(b)$ from the block which represents b. Whether the block which represents $\mu'(b)$ belongs to B or E/D is then determined in order to select the step of Algorithm 3.1 which is to be performed next. However, if $u_1 = 1$, the display processor identifies $\mu'(b)$ by simply executing a jump command.

The average time $\bar{g}_x$ during which the computer is devoted to the

picture generation process while the second component of an element of B is identified also depends on whether the computer or the display processor performs Algorithm 3.1. If the computer performs this algorithm and the blocks which represent the elements of each set P(D[e]) are linked together in a ring with the block which represents D[e] , an explicit reference to the second component of each element of B need not be stored in the block which represents that element. If T_s is the time which is required to transfer a subroutine call command to the display processor,

$$\bar{g}_x = u_7 (1 - u_1) T_p + u_1 T_s + (1 - u_1) (1 - u_7) T_p (1 + |B|/|E/D|) /2. \quad (4.28)$$

The average time $\bar{g}_c$ which is required for the computer to produce $e \in E_0$ from H(e) and $\beta(e)$ is nonzero only if the display processor itself does not perform this task. Furthermore, since a different number of artificial entities may be needed for the structure $\{B, E/D, F'\}$ than for the structure $\{B, E/D, \mu, \gamma\}$, this average time also depends on which form the structure assumes. Then,

$$\bar{g}_c = (1 - u_2) ((1 - u_3) G_{cf} + u_3 G_{cg}). \quad (4.29)$$

where

G_{cf} is the average time which is required for the computer to produce an entity in E_0 for the structure $\{B, E/D, F'\}$, and

G_{cg} is the average time which is required for the computer to produce an entity in E_0 for the structure $\{B, E/D, \mu, \gamma\}$.

(G_{cf} and G_{cg} are estimated from both hardware specifications and the average amount of information which must be converted into display processor format per entity in E_0 for the application under consideration.)

The average time $\bar{g}_t$ during which the computer is suspended while an entity in E_0 is transferred to the display processor depends on the form of the information to be transferred. If the display processor forms $e \epsilon E_0$ from H(e) and $\beta(e)$, $\bar{g}_t$ is the average time which is required to transfer these items to the display processor, whereas, if the computer forms e from H(e) and $\beta(e)$, $\bar{g}_t$ is the average time which is required to transfer the result of this computation to the display processor. Furthermore, since a different number of artificial entities may be needed for the structure $\{B, E/D, F'\}$ than for the structure $\{B, E/D, \mu, \gamma\}$, $\bar{g}_t$ also depends on which structure is used. To summarize,

$$\bar{g}_t = (1 - u_3)((1 - u_2)\, G_{tf} + u_2 G_{tf}') + u_3\, ((1 - u_2) G_{tg} + u_2 G_{tg}'), \qquad (4.30)$$

where

G_{tf} is the average time during which the computer is suspended while an entity which has been formed by the computer is transferred to the display processor for the structure $\{B, E/D, F'\}$,

G_{tf}' is the average time during which the computer is suspended while the subpicture $\beta(e)$ for an entity in $e \epsilon E_0$ is transferred to the display processor for the structure $\{B, E/D, F'\}$,

G_{tg} is the average time during which the computer is suspended while an entity which has been formed by the computer is transferred to the display processor for the structure $\{B, E/D, \mu, \gamma\}$, and

G_{tg}' is the average time during which the computer is suspended while a subpicture $\beta(e)$ for an entity $e \epsilon E_0$ is transferred to the display processor for the structure $\{B, E/D, \mu, \gamma\}$.

(G_{cf} and G_{cg} are estimated from both hardware specifications and the average amount of information which must be transferred to the display processor per entity in E_0 for the application under consideration.)

4.3.2 Time Which Elapses While the Picture is Generated

In order to describe the time t_f which elapses while the picture is generated, the total time which is required for the display processor to plot entities in E_0, exclusive of the time which is required to obtain information which describes these entities from the computer, is considered. Since the number of artificial entities which are present in the picture depends on whether the structure is $\{B, E/D, F'\}$ or $\{B, E/D, \mu, \gamma\}$, this plotting time depends on which structure is used. In further discussion, the plotting time will be denoted T_{df} for the structure $\{B, E/D, F'\}$ and T_{dg} for the structure $\{B, E/D, \mu, \gamma\}$.

Since the display processor must form each entity $e \epsilon E_0$ from $\beta(e)$ and $H(e)$ if $u_2 = 1$, an additional amount of time may be required to transform $\beta(e)$. For example, if an entity in E_0 is described to the display processor as a translation transformation and a list of relative vectors, the entity may be displayed by first drawing an invisible vector to the coordinates at which the entity is to be displayed and then plotting the vectors relative to the resulting coordinates. No part of the entity can be displayed while the invisible vector is being drawn. However, the invisible vector is usually drawn much more rapidly than the visible parts of the entity, since only the endpoints of this vector must be

determined accurately. In further discussion, the time which is required for the display processor to transform a subpicture into an entity in E_0 will be assumed to be small when compared to the time which is required to plot the entity, and, consequently, it will be ignored.

However, if the display processor performs Algorithm 3.1, the time which elapses while it performs the transformation which is described by a value of F', G', or G'' cannot be ignored. Several of these transformations are generally performed in order to produce a single entity in E_0. If the structure is $\{B, E/D, F'\}$, and $t_{tf}(\rho(e,e'))$ denotes the additional time to that which is required to access memory which is required to perform the transformation which is represented by $F'(\rho(e,e'))$, the total time T_{tf} which is consumed in this process is

$$T_{tf} = 2 \sum_{(e,e')\in A} \psi(e)\, t_{tf}(\rho(e,e')). \tag{4.31}$$

(The factor 2 is included in this equation to account for the fact that the transformations represented by both $F'(\rho(e,e'))$ and $F'(\rho(e,e'))^{-1}$ must be performed $\psi(e)$ times.) If the structure is $\{B, E/D, \mu, \gamma\}$, the time T_{tg} which is required to perform the transformations which are represented by values of G' and G'' is much more difficult to express for the general application. However, if the form of the structure is known, T_{tg} may be expressed much more easily. Specific values of T_{tg} will be estimated for several examples in Chapter 5 where they are needed to optimize implementations of the structure.

In many hardware configurations, the display processor cannot suspend the computer continuously by accessing its memory continuously. For example, the display processor may be interfaced to the computer so that it can use only every other cycle. In this event, the total time g during which the computer is suspended while the picture is generated is less than the total time that elapses while the display processor accesses memory in order to perform Algorithm 3.1. However, the time which elapses during this process is generally proportional to g. Consequently, the factor σ is defined to be the ratio of the time which elapses while the display processor accesses memory to the time during which the computer is suspended while the display processor accesses memory. For example, if the display processor can use only every other memory cycle, $\sigma = 2$. This factor is used below to help describe t_f.

If the computer performs Algorithm 3.1, this algorithm may be performed while entities in E_0 are being plotted by the display processor. If the display processor has a vector drawing capability, the time which is required to plot each entity $e' \in E_0$ is generally greater than the time which is required to compute H(e) and to determine $\beta(e)$ for any other entity $e \in E_0$. Then, H(e) and $\beta(e)$, where $e \in E_0$ is the next entity in E_0 to be displayed after $e' \in E_0$ is displayed, may be determined while e' is being plotted by the display processor, and t_f is the total time which

is required to plot all entities in E_0. By contrast, if the display processor must perform Algorithm 3.1 itself, the computation of H(e) and β(e) for $e \epsilon E_0$ cannot be overlapped with the plotting of an entity $e' \epsilon E_0$. Consequently, for this type of operation, t_f is the sum of the total plotting time for entities in E_0 and the time which elapses while values of H and β are computed for these entities. As will be described in Section 4.5, g will always be assumed to be less than the total plotting time for entities in E_0 if $u_1 = 0$. If this condition is assumed to imply that the time which is required for the computer to determine H(e) and β(e) for each entity $e \epsilon E_0$ is less than the time which is required to plot this entity, t_f may be expressed as follows:

$$t_f = (1 - u_3)T_{df} + u_3 T_{dg} + u_1 (g\sigma + (1 - u_3) T_{tf} + u_3 T_{tg}). \qquad (4.32)$$

4.4 Effect of Design Decisions on Response Time

As has been described in Chapter 3, many of the basic operations may be implemented in a variety of ways. These various methods of implementation are described by various sets of values for the design decisions. Consequently, the average time which is required to perform each basic operation is a function of the design parameters. In addition, the frequency of occurrence of each basic operation is a function of the design parameter u_3, i.e.,

$$f(\ell_i) = (1 - u_3) f_f(\ell_i) + u_3 f_g(\ell_i), \quad i = 1, 2, \ldots, 15, \tag{4.33}$$

where

$f_f(\ell_i)$ is the frequency of occurrence of Operation ℓ_i for the structure $\{B, E/D, F'\}$, and

$f_g(\ell_i)$ is the frequency of occurrence of Operation ℓ_i for the structure $\{B, E/D, \mu, \gamma\}$.

(Note that $f_f(\ell_8) = f_g(\ell_{13}) = f_f(\ell_{14}) = f_f(\ell_{15}) = 0$.) The average time which is required to perform each basic operation is expressed as a function of the design decisions below.

4.4.1 Operations Which Modify the Structure

The average time $\bar{r}_1$ which is required to synchronize with the picture generator and to store references to a newly created element of B and the average time $\bar{r}_2$ which is required to synchronize with the picture generator and remove references to an element of B which is to be destroyed depend on the values of the design parameters. The possible references to an element $\rho(e, e') \in B$ which must be changed when that element is created or destroyed are the following:

(1) A reference to $\rho(e, e')$ which is stored in the block which represents $\mu(\rho(e, e'))$. This reference is always present, since it is needed to generate the picture.

(2) A reference to $\rho(e, e')$ which is stored in the block which

represents $\mu'(\rho(e,e'))$. If $\mu'(\rho(e,e'))\epsilon E/D$, this reference is stored only if $u_9 = 1$, if $\mu'(\rho(e,e'))\epsilon B$, this reference is stored only if $u_8 = 1$.

(3) References to $\rho(e,e')$ which are stored in the elements of the ring which represents $P(D[e'])$. These references exist only if $u_{10} = 1$, and the number of them is determined by the value of u_{11}.

The first of these references may be stored as either a pointer or as a display processor jump command, depending on whether the computer or the display processor performs Algorithm 3.1. The time which is required to store this reference in the structure is then

$$R_\mu' = (1 - u_1) R_p + u_1 R_j, \tag{4.34}$$

where

R_p is the time which is required to store a pointer, and

R_j is the time which is required to store a display processor jump command.

If a reference to $b\epsilon B$ is stored in the block which represents $\mu'(b)$, it is stored as a pointer, since it is not interpreted by the display processor. Whether or not this reference is stored for a particular element $b\epsilon B$ may depend on whether $\mu'(b)$ belongs to B or E/D. Since there are $|E/D| - |E_0/D_0|$ elements $b\epsilon B$ for which $\mu(b)\epsilon E/D$, the probability that $\mu(b)\epsilon B$ is $(|B| - |E/D| + |E_0/D_0|)/|B|$. The average time which is required to store this reference is then

$$R_{\mu} = u_9 R_p (|E/D| - |E_0/D_0|)/|B| + u_8 R_p (|B| - |E/D| + E_0/D_0|)/|B| \quad (4.35)$$

The time which is required to add an element $b \in B$ to P(D[e]) or to remove it from P(D[e]) depends on whether both forward and backward pointers or just forward pointers are stored in the ring which represents P(D[e]). The time which is required to store references to a new element of B in the ring which represents P(D[e]) is

$$R_a = u_{10} T_p + (u_{10} + u_{11}) R_p. \quad (4.36)$$

Since the order in which blocks appear in the ring is not interpreted, the new block may be inserted at any position in the ring. So that no part of the ring need be searched, the block is assumed to be inserted in the ring after the block which represents D[e] . A time T_p is then required to obtain the forward pointer from the block which represents D[e] so that it may be stored in the new block. Once this pointer is obtained, $(u_{10} + u_{11})$ pointers to the new block are stored in the ring.

The average time which is required to remove a block from the ring which represents a set P(D[e]) is

$$R_r = u_{10} (1 - u_{11}) (R_p + (1 + |B|/|E/D|) (T_p + T_c)) + 2u_{11} (T_p + R_p). \quad (4.37)$$

If $u_{10} = 0$ (and, consequently, $u_{11} = 0$), the ring does not exist, and $R_r = 0$. If only forward pointers are stored in the ring ($u_{10} = 1$ and $u_{11} = 0$), the entire ring must be searched to find the block which contains

a pointer to the block to be removed. The average number of blocks in this ring is $1 + |B|/|E/D|$. Each time that a pointer is examined during the search (which requires the time T_p), it must be compared with the address of the block to be removed (which requires the time T_c) in order to determine whether or not the search should be terminated. When the search is complete, the pointer to the block to be removed is changed to the forward pointer which is stored in the block to be removed. If both forward and backward pointers are stored in the ring (i.e., $u_{10} = u_{11} = 1$), the forward and backward pointers in the block to be removed are examined to identify the blocks which contain pointers to the block to be removed. The pointers to the block to be removed which are stored in these blocks are then changed.

The average time which is required to synchronize with the picture generator and store references to a newly created element of B is then

$$\bar{r}_1 = R_\mu + R_\mu' + R_a + u_3 t_f/2. \tag{4.38}$$

(Recall from Chapter 3 that the program which is modifying the structure must wait for Algorithm 3.1 to be completed only if the structure is $\{B, E/D, \mu, \gamma\}$ in order to store references to a newly created element of B.) Similarly, the average time which is required to synchronize with the picture generator and remove references to an element of B from the structure is

$$\bar{r}_2 = R_\mu + R_\mu' + R_r + t_f/2. \tag{4.39}$$

(Recall from Chapter 3 that the program which is modifying the structure must always wait for Algorithm 3.1 to be completed in order to modify references to an element of B which is to be destroyed.) Then, from these equations and Equations 3.38 and 3.39, the average

times which are required to perform operations ℓ_3 and ℓ_4 (which create and destroy elements of B) are the following:

$$\bar{r}_{\ell}(\ell_3) = (\bar{r}_g + r_c s_b)\, t_f/(t_f - g) + \bar{r}_{\ell}(\ell_{10}) + R_{\mu} + R_{\mu}' + R_a + u_3\, t_f/2, \tag{4.40}$$

and

$$\bar{r}_{\ell}(\ell_4) = \bar{r}_d t_f/(t_f - g) + \bar{r}_{\ell}(\ell_9) + \bar{r}_{\ell}(\ell_{10}) + R_{\mu} + R'_{\mu} + R_r + t_f/2. \tag{4.41}$$

The average times which are required to perform the basic operations ℓ_1 and ℓ_2 (which create and destroy elements of E/D) have been described by Equations 3.36 and 3.37. The average time which is required to perform operation ℓ_{15} (which modifies the second component of an element of B in the structure $\{B, E/D, \mu, \gamma\}$) has been described in Chapter 3 as being approximately $t_f/2$. Because $t_f/2$ is very large when compared to the time which is required to modify a reference, no attempt will be made to estimate $\bar{r}_{\ell}(\ell_{15})$ more precisely, i.e.,

$$\bar{r}_{\ell}(\ell_{15}) = t_f/2. \tag{4.42}$$

4.4.2 Operations Which Interrogate the Structure

As described in Chapter 3, Operation ℓ_5, which of a set P(D[e]), may be performed by examining a ring (if $u_{10} = 1$), by Algorithm 3.5, or by waiting for the picture generator to identify

these elements. As has been described, the average time which is required to perform Algorithm 3.5 is difficult to express for the general application. However, the design parameters on which it depends can be easily specified. Since the number of times that each step of the algorithm is performed depends on the cardinalities of the sets B, $E/D - E_0/D_0$ and E_0/D_0, the time which is required to perform the algorithm depends on u_3. Furthermore, since references to second components of elements of B are examined by the algorithm, the time which is required to perform the algorithm depends on the way in which these references are stored, which is described by u_1 and u_7. Then, by recalling Equation 3.41 and the fact that the program which is modifying the structure must wait a time $3t_f/2$ in order to perform ℓ_5 if the picture generator identifies the desired elements, $\bar{r}_\ell(\ell_5)$ may be expressed as follows:

$$\bar{r}_\ell(\ell_5) = (u_{10}(T_p + T_c)(1 + |B|/|E/D|) + u_4((1 - u_3)((1 - u_1)(u_7 R_{sf} + (1 - u_7) R_{sf}') + u_1 R_{sf}'') + u_3((1 - u_1)(u_7 R_{sg} + (1 - u_7) R_{sg}') + u_1 R_{sg}''))t_f/(t_f - g) + 3(1 - u_4)(1 - u_{10}) t_f/2, \quad (4.43)$$

where

R_{sf} is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, F'\}$ with $u_1 = 0$ and $u_7 = 1$,

R_{sf}' is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, F'\}$ with $u_1 = u_7 = 0$,

R_{sf}'' is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, F'\}$ with $u_1 = 1$,

R_{sg} is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, \mu, \gamma\}$ with $u_1 = 0$ and $u_7 = 1$,

R_{sg}' is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, \mu, \gamma\}$ with $u_1 = u_7 = 0$, and

R_{sg}'' is the average time which is required to perform Algorithm 3.5 for the structure $\{B, E/D, \mu, \gamma\}$ with $u_1 = 1$.

As has been described in Chapter 3, operations ℓ_6 and ℓ_{10} each require the same amount of time to perform. Each of these operations involves examining a display processor jump command if $u_1 = 1$, or a pointer if $u_1 = 0$. Consequently, if T_j' denotes the time which is required to identify the block which a display processor jump command references,

$$\bar{r}_{\ell}(\ell_6) = \bar{r}_{\ell}(\ell_{10}) = ((1 - u_1)\ T_p + u_1\ T_j')\ t_f/(t_f - g) \qquad (4.44)$$

The time which is required to perform Operation ℓ_7 (which identifies $b \epsilon B$ for a given $\mu'(b) \epsilon E/D$) depends on the value of u_9. If $u_9 = 1$, this operation involves the examination of a pointer, whereas, if $u_9 = 0$, Operation ℓ_{10} must be performed several times in succession. Consequently, from Equations 3.44 and 3.45,

$$\bar{r}_\ell(\ell_7) = u_9 T_p t_f/(t_f - g) + (1 - u_9)(\bar{r}_\ell(\ell_6) + (\bar{r}_\ell(\ell_{10}) + T_c t_f/(t_f - g))|B|/(|E/D| - |E_0/D_0|)). \quad (4.45)$$

Similarly, Operation ℓ_9 (which identifies $b \epsilon B$ for a given $\mu'(b) \epsilon B$) depends on the value of u_8. From Equations 3.46 and 3.48,

$$\bar{r}_\ell(\ell_9) = u_8 T_p t_f/(t_f - g) + (1 - u_8)(\bar{r}_\ell(\ell_{11}) + (\bar{r}_\ell(\ell_{10}) + T_c t_f/(t_f - g))(1 + |B|/(|E/D| - |E_0/D_0|))/2). \quad (4.46)$$

The time which is required to retrieve transformation matrices from the structure depends on the format in which these matrices are stored. Generally, this format is determined by which device, the computer or the display processor, performs Algorithm 3.1. Consequently,

$$\bar{r}_\ell(\ell_8) = ((1 - u_1) R_g + u_1 R_g') t_f/(t_f - g) \quad (4.47)$$

where

R_g is the average time which is required to retrieve the matrix $G(D[e])^{-1}$ from the structure for a class $D[e] \epsilon E_0/D_0$ if the computer performs Algorithm 3.1, and

R_g' is the average time which is required to retrieve this matrix if the display processor performs Algorithm 3.1.

Similarly,

$$\bar{r}_{\ell}(\ell_{13}) = ((1 - u_1) R_f + u_1 R_f') t_f / (t_f - g) \qquad (4.48)$$

where

R_f is the average time which is required to retrieve the matrix $F'(\rho(e,e'))$ from the block which represents $\rho(e,e')$ if the computer performs Algorithm 3.1, and

R_f' is the average time which is required to retrieve this matrix if the display processor performs Algorithm 3.1.

The time which is required to determine the form of $\gamma(\rho(e,e'))$ by examining the block which represents $\rho(e,e')$ also is determined by the format in which this sequence is represented. This format is determined by which device performs Algorithm 3.1, i.e.,

$$\bar{r}_{\ell}(\ell_{14}) = ((1 - u_1) R_{\gamma} + u_1 R_{\gamma}') t_f / (t_f - g) \qquad (4.49)$$

where

R_{γ} is the time which is required to determine the form of $\gamma(b)$ if the computer performs Algorithm 3.1, and

R_{γ}' is the time which is required to determine the form of $\gamma(b)$ if the display processor performs Algorithm 3.1.

The time which is required to perform Operation ℓ_{11} (which identifies the first component of an element of B) depends on whether

or not a reference to the first component of each element of B is stored in the block which represents that element. If this reference is stored, it is examined to perform ℓ_{11}; otherwise, Algorithm 3.8 must be applied. From Equations 3.49 and 3.50, the average time which is required to perform this operation is then

$$\bar{r}_{\ell}(\ell_{11}) = u_6 T_p t_f/(t_f - g) + (1 - u_6)(\bar{r}_{\ell}(\ell_{10}) + T_b t_f/(t_f - g))(1 + |B|/(|E/D| - |E_0/D_0|))/2). \quad (4.50)$$

The time which is required to perform Operation ℓ_{12} (which identifies the second component of an element of B) depends on whether or not an explicit reference to the second component of each element of B is stored in the block which represents that element. This reference may be stored as either a pointer or a display processor subroutine call command, depending on the value of u_1. From Equations 3.51, 3.52, and 3.53, the average time which is required to perform this operation is

$$\bar{r}_{\ell}(\ell_{12}) = u_7((1 - u_1) T_p + u_1 T_s')\, t_f/(t_f - g) + (1 - u_7)((T_p + T_b)\, t_f/(t_f - g))(1 + |B|/|E/D|)/2. \quad (4.51)$$

(Recall that T_s' denotes the time which is required to identify the block which a display processor subroutine call references.)

4.5 Constraints

Because there are eleven design parameters, each of which may

assume two values, the number of implementations of the topological structure which can be specified for a given hardware configuration is $2^{11} = 2048$. However, only 480 of these implementations are possible because the design parameters are not independent. Furthermore, for certain applications, not all of these 480 implementations are possible. The constraints on the design parameters which result from the way that these parameters were defined are described below. The constraints which are imposed on the design parameters by the hardware configuration and the application are then described.

The constraints which may be derived from the definitions of the design parameters are the following:

$$u_3 = 1 \Rightarrow u_5 = 0 \tag{4.52}$$

$$u_7 = 0 \Rightarrow u_{10} = 1 \text{ and } u_1 = 0 \tag{4.53}$$

$$u_{10} = 1 \Rightarrow u_4 = 0 \tag{4.54}$$

$$u_{10} = 0 \Rightarrow u_{11} = 0 \tag{4.55}$$

Equation 4.52 states that matrices cannot be saved on the push-down stack if the structure is $\{B, E/D, \mu, \gamma\}$. This constraint is true because Step 4 of Algorithm 3.1 does not necessarily restore J to a value which it assumed previously for this structure. Consequently, nothing can be achieved by saving matrices on the push-down stack.

Equation 4.53 states that each set P(D[e]) must be represented as a ring in the structure so that the second component of each element of B may be identified if a reference to this component is not stored in each block which represents an element of B. However, the display processor is assumed to be incapable of identifying this component from this ring. Consequently, this equation also states that the display processor must not perform Algorithm 3.1 if no explicit reference to the second component of each element of B is stored in the block which represents that element. If each set P(D[e]) is represented as a ring in the structure, the best way to perform Operation ℓ_5 is to examine this ring. Consequently, Equation 4.54 states that ℓ_5 is not performed by Algorithm 3.5 if each set P(D[e]) is represented in this way. Equation 4.55 states that both forward and backward pointers are not stored in the ring which represents a set P(D[e]) if this ring does not exist.

Other constraints which depend on the application of the structure are limitations on the values which s and t_f are allowed to assume. If the picture generator program (which is needed if $u_1 = 0$) occupies an appreciable amount of storage, the amount of storage which is available may be a function of u_1. However, the picture generator program generally occupies much less storage than the topological structure, and, consequently, the error introduced by assuming that

the maximum available storage for the structure is independent of u_1 is not significant. Then, if s_{max} denotes the amount of storage which is available to represent the structure and t_{max} denotes the amount of time which is available to generate the picture, these constraints may be written as follows:

$$s \leq s_{max} \tag{4.56}$$

and

$$t_f \leq t_{max} \tag{4.57}$$

However, these constraints do not completely describe the limitations which are imposed by the application. In addition to these limitations, the computing time which is available to operate on the structure may be limited by the time during which the computer must be devoted to the picture generation process. In deriving Equation 3.34, the picture was assumed to be continually generated, even while the response to a control language input is being processed. Then, if $g \geq t_f$, the computer must be completely devoted to the picture generation process, and no time may be spent responding to control language inputs. Consequently, a finite response time is achieved only if

$$g < t_f. \tag{4.58}$$

4.6 Conclusion

The number of implementations which may be specified by the design parameters is small enough that these implementations may be

easily enumerated with a computer program. Consequently, the optimum design of the topological structure for a specific hardware configuration and application may be easily determined. A program which enumerates the possible implementations for a specified hardware configuration and application is described in Appendix B. The results obtained by applying this program to several applications for the DEC 339 [1,52] are described in the following chapter.

Chapter 5

APPLICATIONS

The cost function and constraints have been described as functions of eleven design decisions in Chapter 4. The application of the equations stated in that chapter to a given application of the topological structure and a given hardware configuration remains to be demonstrated. In this chapter, two example applications of a DEC 339 [1, 52] (with the extended arithmetic element and VA38 character generator options) are studied through the use of these equations. The results which are given were obtained through the use of the programs which are described in Appendix B.

Because of hardware limitations, the only coordinate transformations which will be considered in these two examples are translations (in two dimensions). The display processor may easily perform these transformations by drawing invisible vectors. Although the DEC 339 display processor does include a scale facility, this facility provides only the very coarse scale factors 1, 2, 4, and 8. Furthermore, vectors which are drawn with scale factors 4 and 8 appear on the screen as dotted lines, whereas those drawn with scale factors 1 and 2 appear as solid lines. For these reasons, coordinate transformations which scale coordinates will not be considered. This limitation of the discussion to only very simple coordinate transformations does not

restrict the values which the design decisions may assume.

One hardware limitation, however, does restrict the values which the design decisions may assume. The DEC 339 display processor is not capable of saving coordinates (i.e., representations of coordinate transformations which are translations) on a push-down stack*, although it is capable of saving references to elements of B. Consequently, the constraint

$$u_1 = 1 \Longrightarrow u_5 = 0 \qquad (5.1)$$

is imposed on the design decisions by the DEC 339 hardware. Furthermore, the matrix J in Algorithm 3.2 cannot be saved on the push-down stack whenever the display processor performs Algorithm 3.1. However, since the only coordinate transformations which are considered are translations, Algorithm 5.1, which is described in Section 5.2.1 below, may be substituted for Algorithm 3.2. This algorithm does not save coordinate transformation matrices on a push-down stack, but it is otherwise very similar to Algorithm 3.2.

5.1 Methods of Implementation

In order to estimate the parameters which describe an application of the topological structure for the DEC 339, the method by which

---

*Rather simple display processors which can save coordinates on a push-down stack have been suggested [44], and at least one modern display processor, LDS-1 [18], can save general transformation matrices on a push-down stack.

certain features of the topological structure are implemented for given values of the design decisions must be specified. For example, if $u_2 = 0$, each subpicture which is associated with a class in E_0/D_0 must be stored as a table of plotting information. However, this table may be stored in any one of several formats. In order to determine the average storage $S_{\beta f}$ or $S_{\beta g}$ which is occupied by one of these subpictures for either value of u_3, the format of this table must be specified.

The formats which will be considered for storing various items are described below in Section 5.1.1. Then, in Section 5.1.2, elementary instruction sequences are given for performing very primitive operations, such as examining a pointer or storing a jump command in the structure. These methods of implementation will be assumed when parameters are estimated later in Section 5.3.

5.1.1 Storage Formats

The items which may be stored to represent a topological structure on a DEC 339 are the following:

(1) a pointer,

(2) a display processor jump command,

(3) a display processor push jump (i.e., subroutine call) command,

(4) a display processor pop (i.e., subroutine return) command,

(5) a display processor internal stop command (i.e., a command which stops the display processor and interrupts the computer),

(6) a pair of coordinates which represents a coordinate transformation matrix,

(7) a subpicture for a class in E_0/D_0,

(8) a value of γ, and

(9) a block type.

Table 5.1 shows the number of locations which each item occupies, as well as the format of each item. The symbols "VEC", "SVEC", and "CHAR" represent the vector mode (1121_8), short vector mode (1141_8), and character mode (1171_8) commands, respectively. The formats for values of γ for $u_1 = 1$ are assumed to be those used by the program which is described in Appendix A. Because of their complexity, these formats are not shown in Table 5.1. Shaded areas in each format are areas into which other items may be packed, whereas unlabeled areas do not contain any useful information. Note that all variable length formats are considered to occupy an integer number of locations so that Algorithm 4.1 may be applied to help compute $\bar{s}_b$, $\bar{s}_{d0}$, and s_d. In addition to the items shown in this table, one additional location is assumed to be needed at each end of each storage block for storage allocation purposes.

5.1.2. Elementary Instruction Sequences

Sequences of instructions which examine and modify these items are shown in Table 5.2, together with the time which is required to perform each sequence. At the beginning of each sequence, location P is assumed to contain a pointer to the first word of each item which is

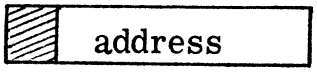
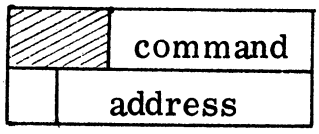
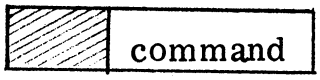
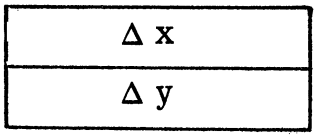
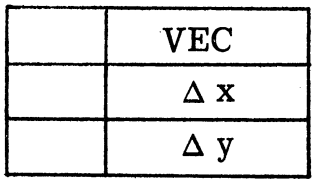
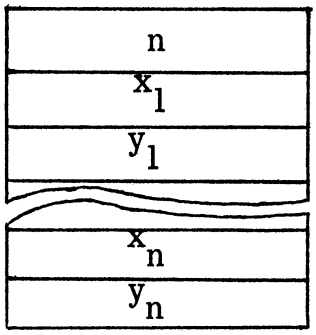
Item	Number of Locations	Format
Pointer	5/6	
Jump or push jump command	5/3	
Pop or internal stop command	2/3	
Coordinate transformation matrix:		
$u_1 = 0$	2	
$u_1 = 1$	3	
Some subpictures for classes in E_0/D_0:		
$u_2 = 0$, n vectors	2n + 1	

Table 5.1

Storage Requirements for DEC 339

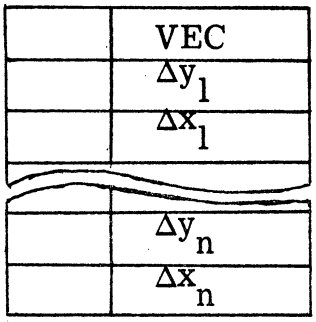
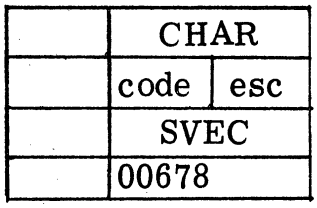
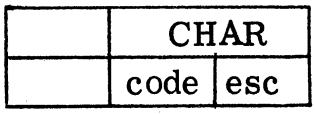
Item	Number of Locations	Format
$u_2 = 1$, n vectors	2n+1	
$u_2 = 1$, $u_3 = 0$, character	4	
$u_2 = 1$, $u_3 = 1$, character	2	
A sequence $\gamma(\rho(e,e'))$:		
$u_1 = 0$	1/9	2 bits
$u_1 = 1$:		
$(I, G(D[e']), I)$	4	
$(I, G(D[e']), G(D[e'])^{-1}$	2	See Appendix A
$(G(D[e'])^{-1}, G(D[e']), I)$	2	
$(G(D[e'])^{-1}, G(D[e']), G(D[e'])^{-1}$	10	
A block type	1/9	2 bits

Table 5.1 (cont.)

Storage Requirements for DEC 339

	Function	Sequence	Time Required
1.	Examine pointer	LAC I P DAC P1	.005 msec
2.	Interpret push jump or jump command as a pointer	ISZ P LAC I P DAC P1	.007 msec
3.	Check block type against given block type	LAC I P AND M SAD C JMP ---- JMP ----	.008 msec
4.	Compare two pointers	LAC P1 SAD P2 JMP ---- JMP ----	.005 msec
5.	Save word on push-down stack	DAC I PS ISZ PS	.005 msec
6.	Restore word from push-down stack	CLA CMA TAD PS DAC PS	.005 msec
7.	Premultiply by value of F' ($u_1 = 0$)	LAC I P TAD X DAC X ISZ P LAC I P TAD Y DAC Y	.016 msec
8.	Premultiply by inverse of value of F' ($u_1 = 0$)	LAC I P CMA TAD (1) TAD X DAC X	.022 msec

Table 5.2

Elementary DEC 339 Instruction Sequences

	Function	Sequence	Time Required
8.	(cont.)	ISZ P LAC I P CMA TAD (1) TAD Y DAC Y	
9.	Copy one word	LAC I P1 DAC I P2	.006 msec
10.	Store pointer in structure	LAC P1 DAC I P	.005 msec
11.	Store jump command in structure	LAC P1 LRSS 14 XOR (2000) DAC I P ISZ P AND (7) LLS 14 DAC I P	.046 msec
12.	Read transformation matrix ($u_1 = 0$)	LAC I P DAC X ISZ P LAC I P DAC Y	.012 msec
13.	Read transformation matrix ($u_1 = 1$)	ISZ P LAC I P JMS S DAC Y ISZ P LAC I P JMS S DAC X ----	.082 msec

Table 5.2 (cont.)

Elementary DEC 339 Instruction Sequences

Function		Sequence		Time Required
13. (cont.)	S,	0		
		ALS	7	
		GSM		
		ALS	1	
		LRS	10	
		SZL		
		TAD	(1)	
		JMP I	S	
14. Read form of $\gamma(\rho(e,e'))$ ($u_1 = 0$)		LAC I	P	.007 msec
		AND	M	
		DAC	T	
15. Read form of $\gamma(\rho(e,e'))$ ($u_1 = 1$)		CLA CMA		.012 msec for
		TAD	P	(I, G(D[e']), I)
		DAC	P	
		CMA		.014 msec for
		TAD I	P	$(G(D[\ e'\ ])^{-1}, G(D[\ e'\ ]),$
		SAD	C	$G(D[\ e'\ ])^{-1})$
		JMP	----	
		SAD	C1	
		JMP	----	.03 msec
		ISZ	P	otherwise
		ISZ	P	
		LAC I	P	
		ISZ	P	
		ISZ	P	
		CMA		
		TAD I	P	
		SPA		
		JMP	----	
		JMP	----	
16. Read value of F' ($u_1 = 1$)		DZM	X	.012 msec
		DZM	Y	identity matrix
		LAC I	P	
		AND	(7777)	.094 msec
				otherwise

Table 5.2 (cont.)

Elementary DEC 339 Instruction Sequences

	Function		Sequence		Time Required
16.	(cont.)		SAD	(1121)	
			SKP		
			JMP	L	
			Sequence 13		
		L,			
17.	Read $G'(\rho(e,e'))$ or $G''(\rho(e,e'))$ $(u_1 = 0)$		LAC	I P	.006 msec
			AND	M	
			SNA		
			JMP	----	
18.	Convert table of n coordinate pairs into vector mode coordinates $(u_2 = 0)$		LAC	I P	(.007+.017n) msec
			DAC	T	
			ISZ	P	
		L,	LAC	I P	
			GSM		
			SZL		
			TAD	(2001)	
			DAC	I P1	
			ISZ	P	
			ISZ	P1	
			ISZ	T	
			JMP	L	

Table 5.2 (cont.)

Elementary DEC 339 Instruction Sequences

to be examined or modified. The contents of other locations which appear in these sequences are interpreted as follows:

PS: pointer to the last item on a push-down stack.

M: mask used to clear selected bits of the accumulator.

C, C1: masks against which the content of the accumulator is to be compared.

P1, P2: pointers.

X, Y: x and y coordinate values.

T: other forms of data.

All arithmetic is assumed to be two's complement arithmetic. A storage allocation scheme similar to that shown in Figure 3.1 is also assumed. Sequence 15 in the table examines the overhead information which is needed for this allocation scheme in order to determine the length of a storage block. This length is then used to help determine the form of the value of γ which is associated with that block.

5.2 Time Required to Perform Algorithms 3.2 and 3.5

Although Algorithms 3.2 and 3.5 have been described in Chapter 3, the time which is required to perform each of these algorithms has not been discussed. However, the time which is required to perform Algorithm 3.2 (which computes a matrix $G(D[e])^{-1}$) must be estimated in order to estimate the parameters G_{mg}, G_{mg}', G_{mg}'', G_{hg}, G_{hg}', and G_{hg}'', which represent average times necessary to premultiply a matrix by values of G' and G'' under various conditions. Similarly,

the time which is required to perform Algorithm 3.5 (which may be used to perform Operation ℓ_5) must be estimated in order to estimate the parameters R_{sf}, R_{sf}', R_{sf}'', R_{sg}, R_{sg}', and R_{sg}'', which represent the time required to perform this algorithm for various values of the design decisions. Although the time which is required to perform these algorithms is difficult to estimate for the general application of the topological structure, these times are easily estimated if the elements of $B \cup E/D$ which are examined by the algorithms can be identified. These elements generally can be identified once the application of the structure is known, as will be demonstrated in Section 5.3.

5.2.1 Time Required to Compute $G(D[e])^{-1}$

The operations which are performed by Algorithm 3.2 may be summarized as follows:

For each $D[e'] \in E_0/D_0$ examined:

(1) Determine that $D[e']$ is an element of E_0/D_0 (Step 2).

(2) Read $G(D[e'])^{-1}$ (Step 2).

For each $D[e'] \in E/D - E_0/D_0$ examined:

(1) Determine that $D[e']$ is an element of $E/D - E_0/D_0$ (Step 2).

(2) Identify $b \in B$ such that $\mu(b) = D[e']$ (Step 2).

For each $b \in B$ examined:

(1) Read form of $\gamma(b)$ so that $G'(b)$ may be compared with $G''(b)$ (Step 3).

(2) Identify $\mu'(b)$ (Step 4).

(3) Determine whether or not $\mu'(b) \in B$ (Step 4).

For each $b=\rho(e', e'') \in B$ examined with $G'(b)=G''(b)$:

(1) Save b with matrix J on push-down stack (Step 3).

(2) Restore b and matrix J from push-down stack (Step 5).

(3) Identify D[e''] (Step 3).

(4) Perform matrix multiplication indicated in Step 5.

Consequently, if the elements of $B \cup E/D$ which are examined by Algorithm 3.2 can be identified, the time which is required to compute $G(D[\ e]\ )^{-1}$ by this algorithm can be estimated.

As has been mentioned above, the DEC 339 display processor is not capable of saving coordinate transformation matrices on a push-down stack. Consequently, Algorithm 3.2 cannot be executed by this display processor. However, since the only coordinate transformations which are considered are translations, these transformations are commutative, and the following simplified form of Algorithm 3.2, which does not save coordinate transformation matrices on a push-down stack, may be executed by this display processor.

Algorithm 5.1

Procedure for computing $G(D[\ e]\ )^{-1}$ when all coordinate transformations are commutative.

(1) Let J=I, let X=D[e] , and empty a push-down stack for use in this algorithm.

(2) If there exists an element $b' \in B$ such that $\mu(b')=X$, let $b=b'$ and proceed with Step 4.

(3) If the number of elements b'' on the push-down stack for which $G(b'') \neq I$ is even, let $J=JG(X)^{-1}$. Otherwise, let $J=JG(X)$. Proceed with Step 6.

(4) If $G'(b) \neq G''(b)$, proceed with Step 5. Otherwise, save b on the push-down stack. Let X be the second component of b and proceed with Step 2.

(5) If $\mu'(b) \in B$, let $b=\mu'(b)$ and proceed with Step 4.

(6) If the push-down stack is empty, terminate with $J=G(D[\,e\,])^{-1}$. Otherwise, restore b from the push-down stack and proceed with Step 5. ∎

In order to illustrate this algorithm, the steps which are needed to compute $G(D[\,e\,])^{-1}$ from the structure shown in Figure 3.6 are shown below. Each step is represented by its number and the items which are affected when it is performed.

(1) $J = I$

$X = D[\,e\,]$

(2) $b = \rho(e, e_1)$

(4) Stack contains $\rho(e, e_1)$

$X = D[\,e_1\,]$

(2) $b = \rho(e_1, e_4)$

(4) Stack contains $\rho(e, e_1)$, $\rho(e_1, e_4)$

$X = D[\, e_4]$

(3) $J = G(D[\, e_4])^{-1}$

(6) $b = \rho(e_1, e_4)$; Stack contains $\rho(e, e_1)$

(5) $b = \rho(e_1, e_5)$

(4) Stack contains $\rho(e, e_1)$, $\rho(e_1, e_5)$

$X = D[\, e_5]$

(3) $J = G(D[\, e_4])^{-1}\, G(D[\, e_5])$

(6) $b = \rho(e_1, e_5)$; Stack contains $\rho(e, e_1)$

(5) $b = \rho(e, e_1)$; Stack empty

(5) $b = \rho(e, e_2)$

(5) $b = \rho(e, e_3)$

(4) Stack contains $\rho(e, e_3)$

$X = D[\, e_3]$

(2) $b = \rho(e_3, e_8)$

(4) Stack contains $\rho(e, e_3)$, $\rho(e_3, e_8)$

$X = D[\, e_8]$

(2) $J = G(D[\, e_4])^{-1}\, G(D[\, e_5])\, G(D[\, e_8])$

(6) $b = \rho(e_3, e_8)$; Stack contains $\rho(e, e_3)$

(5) $b = \rho(e_3, e_9)$

(4) Stack contains $\rho(e, e_3)$, $\rho(e_3, e_9)$

(3) $J = G(D[\, e_4])^{-1}\, G(D[\, e_5])\, G(D[\, e_8])\, G(D[\, e_9])^{-1}$

(6) $b = \rho(e_3, e_9)$; Stack contains $\rho(e, e_3)$

(6) $b = \rho(e, e_3)$; Stack empty

Note that the matrix $G(D[\ e]\ )^{-1}$ which is computed by this algorithm differs from that computed by Algorithm 3.2 for the same structure. The matrix computed by Algorithm 5.1 is

$$G(D[\ e]\ )^{-1} = G(D[\ e_4]\ )^{-1} G(D[\ e_5]\ ) G(D[\ e_8]\ ) G(D[\ e_9]\ )^{-1},$$

whereas that computed by Algorithm 3.2 is

$$G(D[\ e]\ )^{-1} = G(D[\ e_4]\ )^{-1} G(D[\ e_5]\ ) G(D[\ e_9]\ )^{-1} G(D[\ e_8]\ ).$$

According to Equation 3.5, the correct result is obtained by Algorithm 3.2. However, if the coordinate transformations which are represented by values of G are commutative, the result obtained by Algorithm 5.1 is equal to this result.

The operations which are performed by Algorithm 5.1 may be summarized as follows:

For each $D[\ e'] \in E_0/D_0$ examined:

(1) Determine that $D[\ e']$ is an element of E_0/D_0 (Step 2).

(2) Postmultiply J by either $G(D[\ e']\ )^{-1}$ or $G(D[\ e']\ )$ (Step 3).

For each $D[\ e'] \in E/D - E_0/D_0$ examined:

(1) Determine that $D[\ e']$ is an element of $E/D - E_0/D_0$ (Step 2).

(2) Identify $b \in B$ such that $\mu(b) = D[\ e']$ (Step 2).

For each $b \in B$ examined:

(1) Read form of $\gamma(b)$ so that $G'(b)$ may be compared with $G''(b)$ (Step 4).

(2) Identify $\mu'(b)$ (Step 5).

(3) Determine whether or not $\mu'(b) \epsilon B$ (Step 5).

For each $\rho(e', e'') \epsilon B$ examined with $G'(\rho(e', e''))=G''(\rho(e', e''))$:

(1) Save $\rho(e', e'')$ on the push-down stack (Step 4).

(2) Restore $\rho(e', e'')$ from the push-down stack (Step 6).

(3) Identify D[e''] (Step 4).

Note that the only difference between these operations and those performed by Algorithm 3.2 is that matrix multiplications are performed differently. The values of G_{mg}, G_{mg}', G_{mg}'', G_{hg}, G_{hg}', and G_{hg}'' which are shown in Tables 5.5 and 5.9 below were estimated as the total time during which the PDP-9 must be devoted to the picture generation process in order to perform these operations for various values of u_1 and u_7.

5.2.2 Time Required to Perform Algorithm 3.5

The time which is required to perform Algorithm 3.5 may be estimated by a similar procedure. The operations which are performed by this algorithm may be summarized as follows:

For each $\rho(e', e'')$ examined:

(1) Save $\rho(e', e'')$ on the push-down stack (Step 2).

(2) Restore $\rho(e', e'')$ from the push-down stack (Step 5).

(3) Identify D[e''] (Step 2).

(4) Compare D[e''] with D[e] to determine whether or not these classes are identical (Step 2).

(5) Identify $\mu'(\rho(e', e''))$ (Step 5).

(6) Determine whether or not $\mu'(\rho(e', e'')) \in B$ (Step 5).

For each $D[e'] \in E/D - \{D[e] \}$ examined:

(1) Let $Y = Y \cup \{D[e'] \}$ (Step 3).

(2) Determine whether or not $D[e'] \in E_0/D_0$ (Step 4).

For each $D[e'] \in E/D - (E_0/D_0 \cup D[e])$ examined:

(1) Find $b \in B$ such that $\mu(b) = D[e']$ (Step 4).

For each $\rho(e', e'') \in B - P(D[e])$:

(1) Determine whether or not $D[e''] \in Y$ (Step 3).

For each $b \in P(D[e])$:

(1) Let $Z = Z \cup \{b\}$ (Step 2.)

The values of R_{sf}, R_{sf}', R_{sf}'', R_{sg}, R_{sg}', and R_{sg}'' which are shown in Table 5.9 below were estimated as the total time which is required to execute PDP-9 instructions which perform these operations. When these parameters were estimated, the sets Y and Z were each assumed to be stored as a list of pointers in consecutive memory locations. The test to determine whether or not a particular element of E/D is an element of Y in Step 3 of the algorithm was assumed to be performed by searching the table which represents Y.

5.3 Optimum Implementations for Specific Applications

The two example applications of the DEC 339 to be considered are (1) a text editor and (2) a graph theory input program. The purpose of the text editor is to display a page of text (which has been previously prepared by some other means) so that both characters and lines may be added to it or removed from it. The purpose of the graph theory input program is to allow the user to draw and modify undirected graphs to be processed by an analysis program. (Some of the applications of such a graph theory program have been discussed by M. S. Wolfberg [76].) The text editor is discussed because of its simplicity and because it shows the merit of the structure $\{B, E/D, \mu, \gamma\}$ for an extreme case. However, this example does not illustrate the design decision u_4, for operation ℓ_5 is never performed, and it does not illustrate the design decision u_2, for characters are assumed to be produced directly by the display processor. Furthermore, $\psi(e) = 1$ for every entity e in this example. The graph theory example, however, illustrates every design decision, as well as certain entities $e \in E$ for which $\psi(e) > 1$.

5.3.1 A Text Editing Program

In order to estimate the parameters which describe the text editor, the form which the topological structure assumes for this application must be specified. The forms of structure which will be

considered are shown in Figure 5.1. Generally, the picture λ is considered to consist of lines of text, each of which consists of several characters. However, the artificial entities $e_0, e_1', e_2', \ldots, e_{n_\ell}', e_1'', e_2'', \ldots, e_{n_\ell}''$, which have been described in Chapter 2, must also be represented in the structure $\{B, E/D, \mu, \gamma\}$. (Although each character is depicted as being in a separate class in E_0/D_0 in this figure, all characters which differ only in their positions on the screen are considered to be represented by one class in E_0/D_0.)

5.3.1.1 Estimation of Parameters

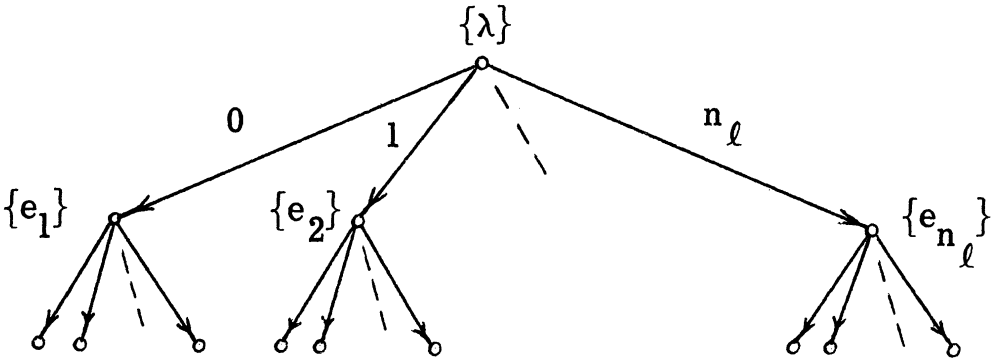
Since the number of entities which are being displayed at the time that a control language input occurs affects the time which is required to respond to that input, the parameters which describe the text editor will be expressed as functions of n_ℓ, the number of lines of text, and n_c, the average number of characters per line. Since there are 64 printing ASCII characters, the number of elements of E_0/D_0 which represent characters will be assumed to be 64. The control language inputs which will be considered are the following:

k_1: Insert a character at a specified position in a specified line .

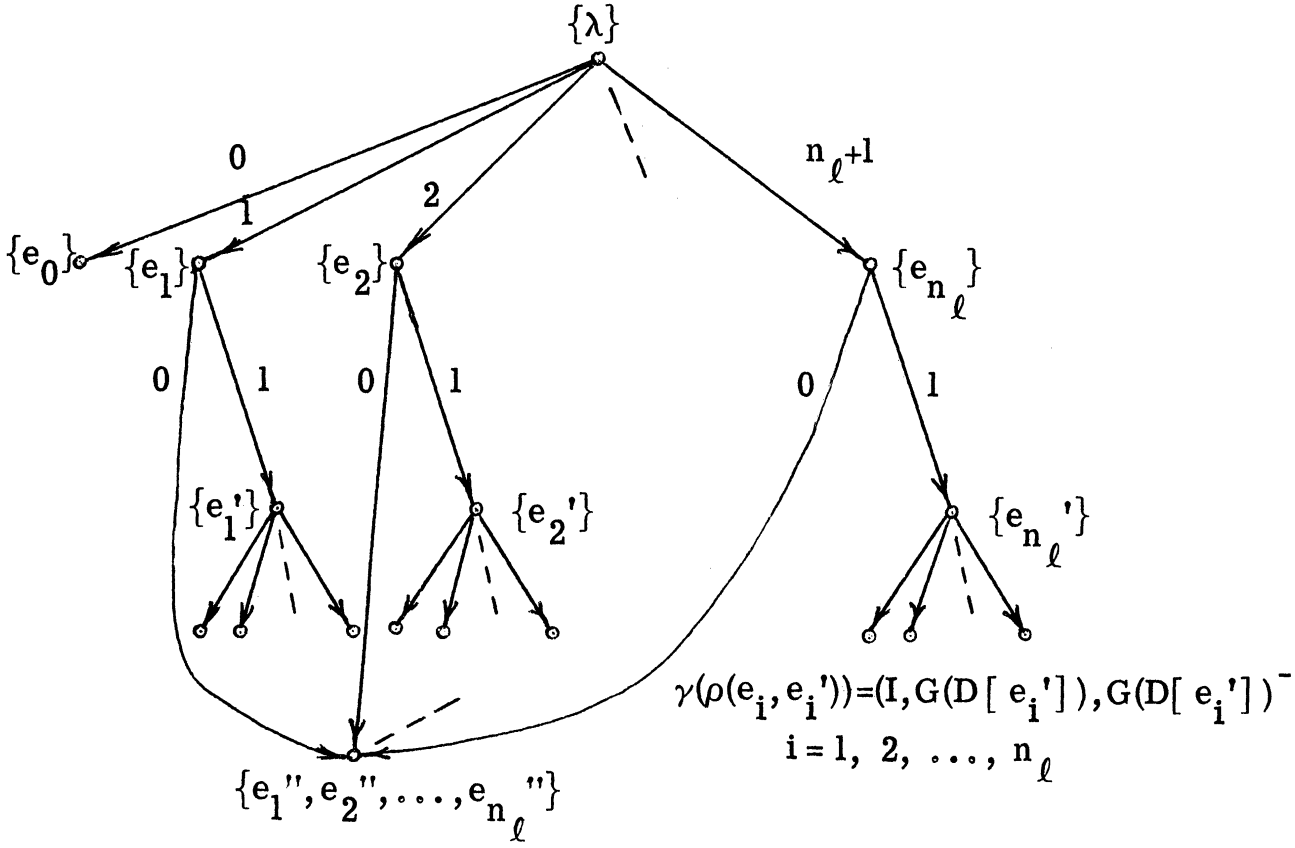
k_2: Insert a line (which contains no characters) at a specified position with respect to other lines.

k_3: Delete a specified character.

k_4: Delete a specified line.



a. $\{B, E/D, F'\}$



b. $\{B, E/D, \mu, \gamma\}$

Figure 5.1

Topological Structures for Text Editing

As will be discussed, none of the responses to these inputs involves the execution of Operation ℓ_5. Consequently, u_4=0, and R_{sf}, R_{sf}', R_{sf}'', R_{sg}, R_{sg}', and R_{sg}'' will not be estimated. All characters will be assumed to be produced by the VA38 character generator, which is part of the display processor. Consequently, u_2=1, and $S_{\beta f}$, $S_{\beta g}$, G_{cf}, G_{tf}, G_{cg}, and G_{tg} will not be estimated. G_s''' is not estimated because the display processor cannot save coordinate transformation matrices on a push-down stack.

Tables 5.3 through 5.5 show values of the parameters for the text editing application. The structure size parameters in Table 5.3 were determined directly from Figure 5.1. The storage parameters in Table 5.4 were determined from Table 5.1. (S_f' is a function of n_ℓ and q_{bf} because the value of F' associated with the first character in each line is I, which is not stored. Furthermore, the storage required for each value of F' which is stored is 6 locations, since the inverse of each value is also assumed to be stored so that the display processor can perform Step 4 of Algorithm 3.1.) The time parameters in Table 5.5 generally were determined from Table 5.2, from hardware specifications, and from the discussion of Algorithms 5.1 and 3.5 in Section 5.2. To facilitate interpretation of this table, references to the applicable instruction sequences from Table 5.2 are included for each entry. The only parameter which is

Parameter	Value
q_{bf}	$n_\ell n_c + n_\ell$
q_{bg}	$n_\ell n_c + 3n_\ell + 1$
q_{df}	$n_\ell + 65$
q_{dg}	$2n_\ell + 67$
q_{d0f}	64
q_{d0g}	66
q_{sf}	$n_\ell n_c + n_\ell$
q_{sg}	$n_\ell n_c + 3n_\ell + 1$
q_{s0f}	$n_\ell n_c$
q_{s0g}	$n_\ell n_c + n_\ell + 1$

Table 5.3

Structure Size Parameters for Text Editor

Parameter	Value (locations)
s_0	2
S_t	1/9
S_t'	1/9
S_p	5/6
S_s	5/3
S_j	5/3
S_a	5/3
S_r	2/3
S_h	2/3
S_f	2
S_f'	$6 - 6n_\ell/q_{bf}$
$S_{\beta f}$	Not estimated
$S_{\beta f}'$	4
S_g	2

Table 5.4

Storage Parameters for Text Editor

Parameter	Value (locations)
S_g'	4
S_γ	1/9
S_γ'	$4 - 2n_\ell/q_{bg}$
$S_{\beta g}$	Not estimated
$S_{\beta g}'$	2.015

Table 5.4 (cont.)

Storage Parameters for Text Editor

Parameter	Value (milliseconds)	Instruction Sequences
T_p	.005	1
T_j	.002	
T_s	.004	
T_j'	.007	2
T_s'	.007	2
T_b	.008	3
T_c	.005	4
T_{tf}	$.0024 n_\ell n_c (n_c - 1) + .1536 n_\ell + .0072 \sum_{i=1}^{n_\ell} \lvert 20 - i \rvert$	
T_{df}	$.02 n_\ell n_c$	
T_{tg}	$.0024 n_\ell n_c$	
T_{dg}	$.02 n_\ell n_c + .0036 n_\ell + .1536$	
G_s	.01	5, 6
G_s'	.003	

Table 5.5

Time Parameters for Text Editor

Parameter	Value (milliseconds)	Instruction Sequences
G_s''	.02	5, 6
G_s'''	Not estimated	
G_{mf}	.016	7
G_{mf}'	$.003-.003\, n_\ell/q_{sf}$	
G_{hf}	.022	8
G_{hf}'	$.003-.003\, n_\ell/q_{sf}$	
G_{mg}	.006	17
G_{mg}'	0	
G_{mg}''	.006	17
G_{hg}	$(.05+.067n_c)\, n_\ell/q_{sg}$	
G_{hg}'	$(.009+.012n_c)\, n_\ell/q_{sg}$	
G_{hg}''	$(.05+.054n_c+.000203125n_\ell n_c)n_\ell/q_{sg}$	
G_{bf}	$(.012\, q_{df} - .004\, q_{d0f})/q_{sf}$	

Table 5.5 (cont.)

Time Parameters for Text Editor

Parameter	Value (milliseconds)	Instruction Sequences
G_{bg}	$(.012q_{dg} - .004q_{d0g})/q_{sg}$	
G_{cf}	Not estimated	
G_{tf}	Not estimated	
G_{tf}'	.004	
G_{cg}	Not estimated	
G_{tg}	Not estimated	
G_{tg}'	$(.003+.002(n_{\ell}+n_{\ell}n_c))/q_{s0g}$	
$\bar{r}_g$	.15	
r_c	.006	9
$\bar{r}_d$	.15	
R_p	.005	10
R_j	.046	11
R_f	.012	12
R_f'	$.094 - .082n_{\ell}/q_{bf}$	16

Table 5.5 (cont.)

Time Parameters for Text Editor

Parameter	Value (milliseconds)	Instruction Sequences
R_g	.012	12
R_g'	.082	13
R_γ	.007	14
R_γ'	$.012+.018n_\ell/q_{bg}$	15
R_{sf}	Not estimated	
R_{sf}'	Not estimated	
R_{sf}''	Not estimated	
R_{sg}	Not estimated	
R_{sg}'	Not estimated	
R_{sg}''	Not estimated	

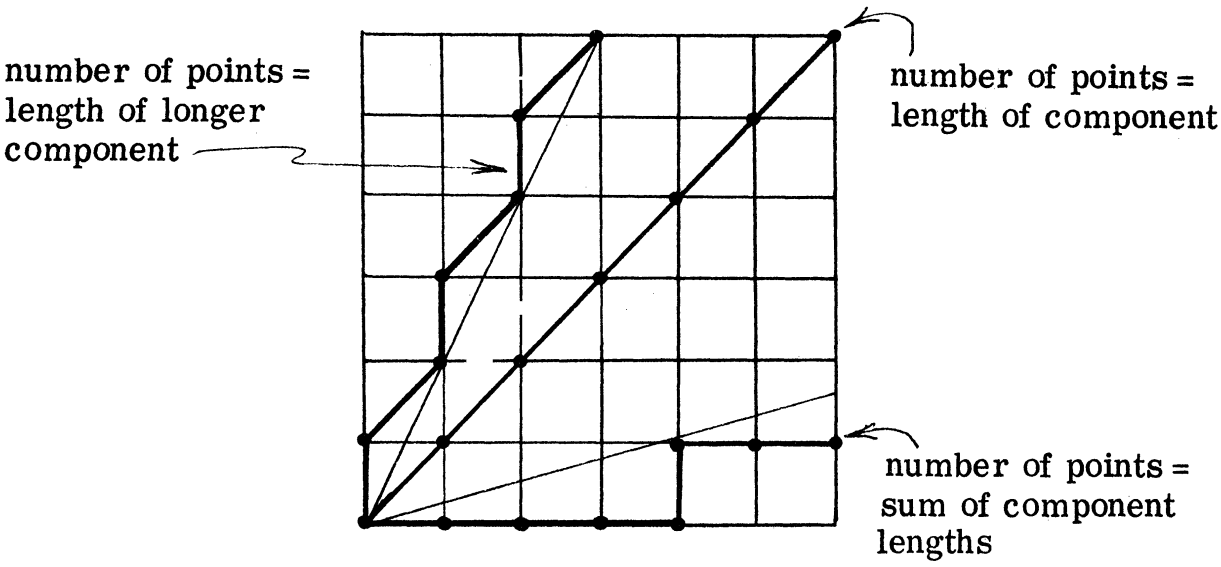
Table 5.5 (cont.)

Time Parameters for Text Editor

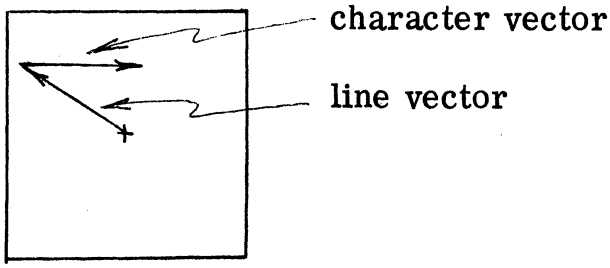
not included in these tables is σ, which has the value 5.

The parameters G_{hg}, G_{hg}', and G_{hg}'' which are shown in **Table 5.5 are** essentially average times required to perform Algorithm 5.1 for various values of u_1 and u_7. The elements of $B \cup E/D$ which are examined by this algorithm are easily identified from Figure 5.1b. From this figure, an inverse of a value of G is seen to be computed once for each line of text, as it is the last component of each $\gamma(\rho(e_i, e_i'))$ for $i=1,2,\ldots,n_\ell$. When Algorithm 5.1 is executed for the line e_i, an average of n_c classes in E_0/D_0 (which contain characters for line e_i) are examined, one class in $E/D-E_0/D_0$ (which contains the entity e_i') is examined, and an average of n_c elements of B (one for each character in line e_i) are examined. Furthermore, for each element $b \in B$ which is examined, $G'(b) = G''(b) = I$.

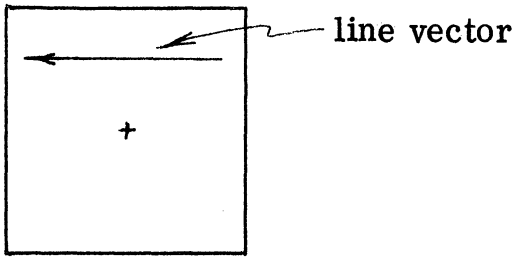
Some of the parameters shown in Table 5.5 were estimated rather coarsely. The storage allocation times $\bar{r}_g$ and $\bar{r}_d$ were estimated by examining storage allocation subroutines which were used in a previous DEC 339 program [27]. The parameters T_{tf} and T_{tg} were estimated by multiplying the total number of points in certain invisible vectors by the plotting time per invisible point. The estimation of the number of points in these vectors is depicted in Figure 5.2. Figure 5.2a shows magnified views of several vectors drawn at different angles. Since each point in each vector can be



a. Types of Vectors



b. Vectors for $\{B, E/D, F'\}$



c. Vectors for $\{B, E/D, \mu, \gamma\}$

Figure 5.2

Estimation of T_{tf} and T_{tg} for Text Editor

situated at one of only a finite number of positions, the number of points in each vector is either (1) the length of the longer component of the vector (in points) or (2) the sum of the lengths of the components of the vector. Since most vectors are drawn at angles which are not rational fractions of π, the number of points in a vector is assumed to be the sum of the lengths of its components. (Note that this assumption introduces no error for horizontal or vertical vectors, although it does introduce error for other vectors which are drawn at angles which are rational fractions of π.) Figures 5.2b and 5.2c show examples of invisible vectors which must be drawn to perform coordinate transformations which correspond to values of F' and inverses of values of G. In Figure 5.2b, the line vector corresponds to the matrix $F'(\rho(\lambda, e_i))$, whereas the character vector corresponds to a value of F' which describes the position of a character in the line e_i. In Figure 5.2b, the line vector corresponds to the matrix $G''(\rho(e_i, e_i')) = G(D[\, e_i'\,])^{-1}$. Since all other values of G' and G'' in the structure shown in Figure 5.1b are identity matrices, these line vectors are the only invisible vectors which were considered when T_{tg} was estimated.

The average number of times that each basic operation is executed in order to respond to each of the inputs k_1, k_2, k_3, and k_4 is shown

in Table 5.6. In order to determine values of the functions f_f and f_g, each input which invokes one of these responses is assumed to be equally likely. The values of these functions are then the averages of the entries in each row in Table 5.6.

5.3.1.2 Results of Analysis

The programs which are described in Appendix B were applied to the parameters and operation frequencies which were estimated above to study various implementations of the structure for various values of n_ℓ and n_c. For each pair of values of n_ℓ and n_c considered, only 48 implementations satisfied the constraint $g<t_f$. The only significant design decision on which the cost was found to depend was u_3. The cost of the best implementation for each value of u_3 is plotted in Figures 5.3 and 5.4. Figure 5.3 shows the cost versus the average number of characters per line for about half the maximum number of lines, whereas Figure 5.4 shows the cost versus the number of lines for about half the maximum number of characters per line. The label on each curve is the set of values of the design decisions . For example, the label 1110001l000 means that $u_1=u_2=u_3=u_7=u_8=1$ and $u_4=u_5=u_6=u_9=u_{10}=u_{11}=0$.

The results which were obtained show clearly that the structure $\{B,E/D,\mu,\gamma\}$ is superior to the structure $\{B,E/D,F'\}$ for this application.

$\{B, E/D, F'\}$

	k_1	k_2	k_3	k_4
ℓ_1	0	1	0	0
ℓ_2	0	0	0	1
ℓ_3	$1+n_c/2$	$1+n_\ell/2$	$n_c/2$	$n_\ell/2$
ℓ_4	$n_c/2$	$n_\ell/2$	$1+n_c/2$	$1+n_c+n_\ell/2$
ℓ_5	0	0	0	0
ℓ_6	0	0	0	1
ℓ_7	0	0	0	0
ℓ_8	0	0	0	0
ℓ_9	0	0	0	0
ℓ_{10}	$n_c/2$	$n_\ell/2$	$n_c/2$	$n_c+n_\ell/2$
ℓ_{11}	0	0	0	0
ℓ_{12}	$n_c/2$	$n_\ell/2$	$n_c/2$	$n_\ell/2$
ℓ_{13}	$n_c/2$	$n_\ell/2$	$n_c/2$	$n_\ell/2$

Table 5.6

Values of the Function $\bar{n}$ for Text Editor

	k_1	k_2	k_3	k_4
ℓ_{14}	0	0	0	0
ℓ_{15}	0	0	0	0

$\{B, E/D, \mu, \gamma\}$

	k_1	k_2	k_3	k_4
ℓ_1	0	2	0	0
ℓ_2	0	0	0	2
ℓ_3	1	3	0	0
ℓ_4	0	0	1	$3+n_c$
ℓ_5	0	0	0	0
ℓ_6	0	0	0	1
ℓ_7	0	0	0	0
ℓ_8	0	0	0	0
ℓ_9	0	0	0	0

Table 5.6 (cont.)

Values of the Function $\bar{n}$ for Text Editor

	k_1	k_2	k_3	k_4
ℓ_{10}	0	0	0	n_c
ℓ_{11}	0	0	0	0
ℓ_{12}	0	0	0	0
ℓ_{13}	0	0	0	0
ℓ_{14}	0	0	0	0
ℓ_{15}	0	0	0	0

Table 5.6 (cont.)

Values of the Function $\bar{n}$ for Text Editor

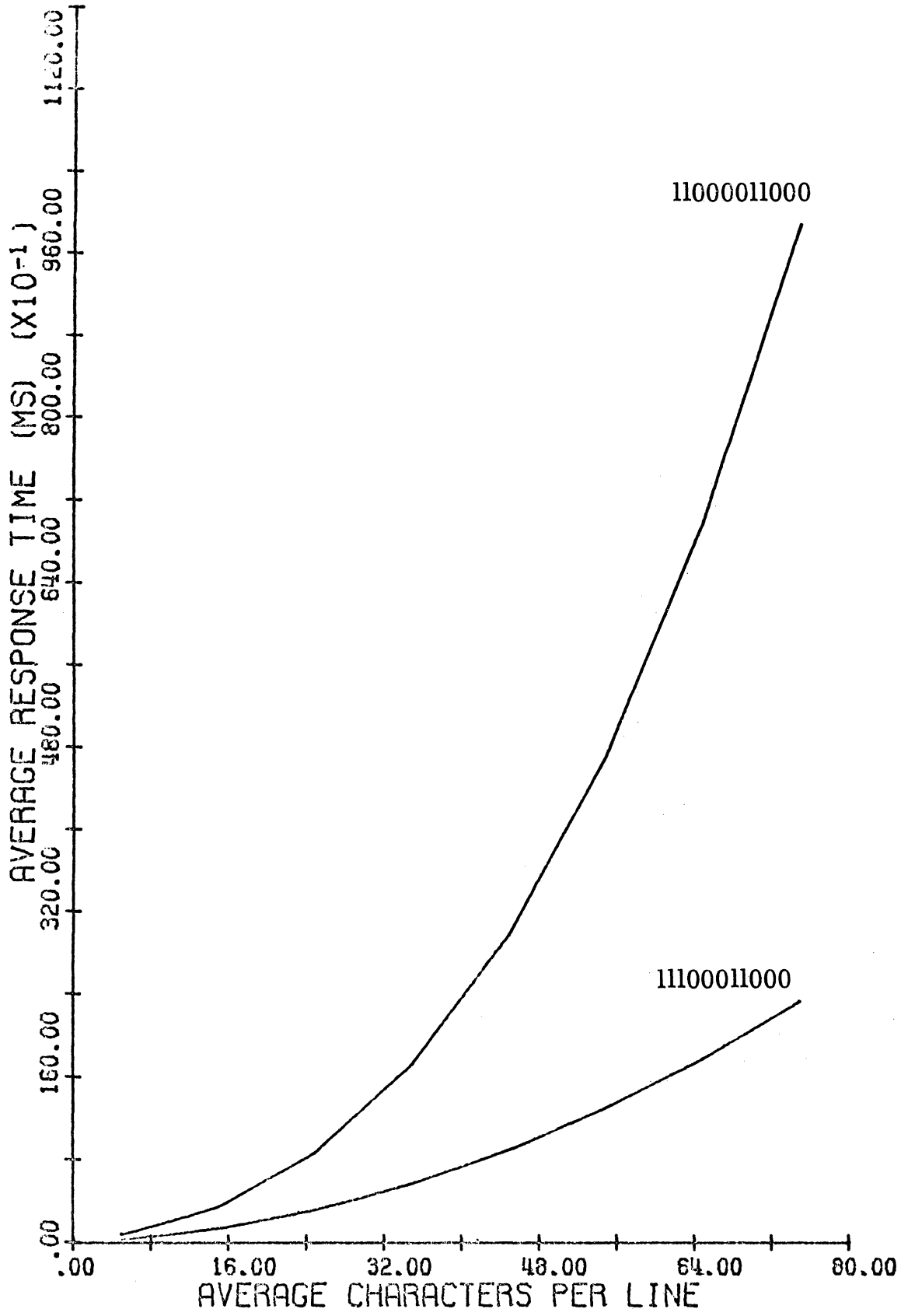


Figure 5.3

Text Editor Cost for $n_\ell = 20$

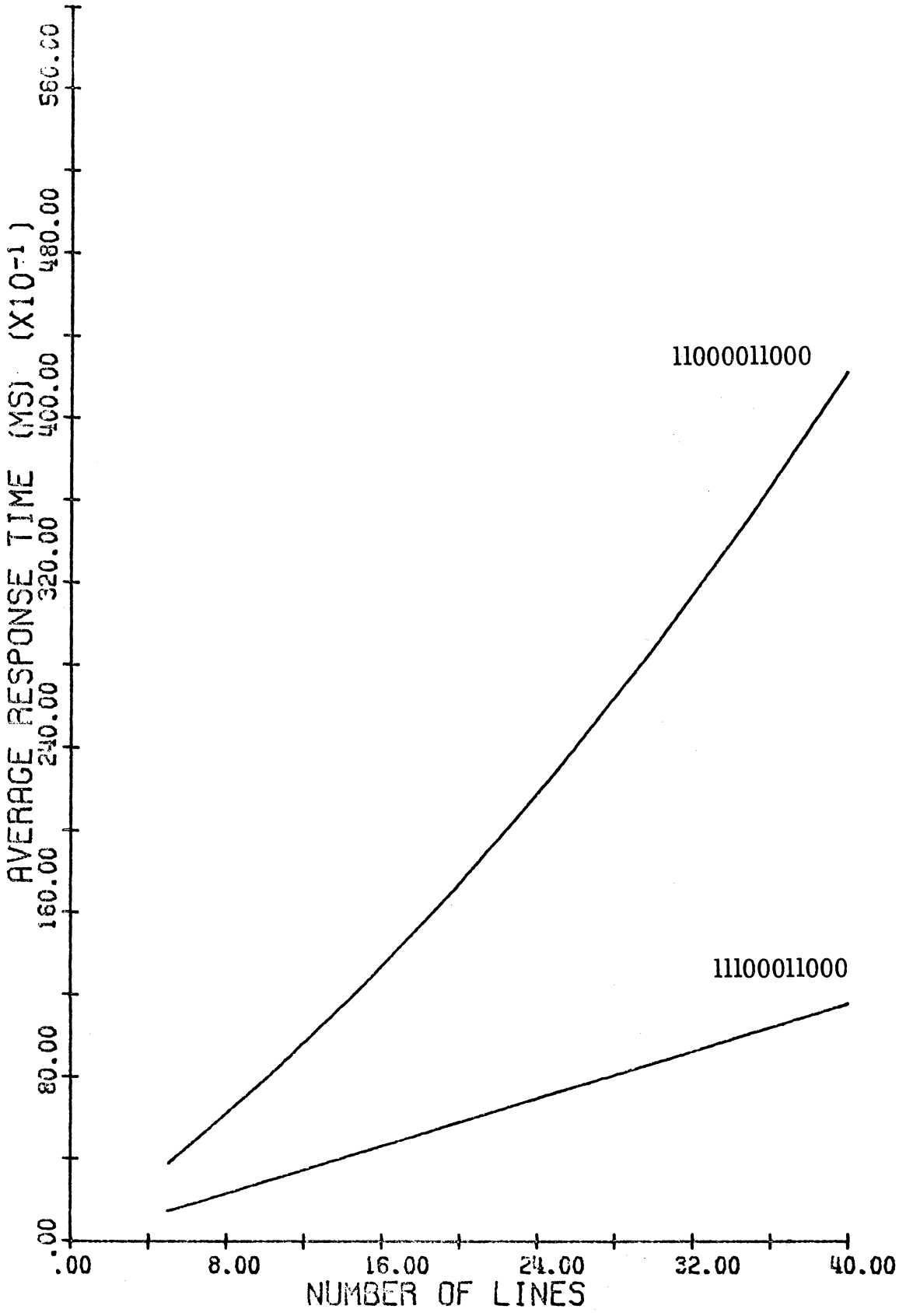


Figure 5.4

Text Editor Cost for $n_c = 35$

The optimum implementation 1110001l000 whose cost is shown in Figures 5.3 and 5.4 requires less storage than the implementation 1100001l000, which is the implementation with u_3=0 which requires the smallest amount of storage. Furthermore, the maximum value of t_f for the implementation 1110001l000 is less than or equal to the maximum value of t_f for any other implementation. Consequently, regardless of the maximum values which the storage s and the elapsed time t_f are allowed to assume, $\{B, E/D, \mu, \gamma\}$ is the better structure for this application of the DEC 339. However, if only a small amount of storage is available, the implementation 1110001l000 may not be possible. In this event, the implementation 1110001l000, which differs from 1110001l000 only in that values of μ are not stored, is the optimum implementation. (The cost of this implementation is only slightly larger than that for 1110001l000.) If insufficient storage is available for this implementation, the text editor cannot be implemented.

Because $f_f(\ell_5) = f_g(\ell_5) = 0$, u_4 was constrained to be zero when this application was analyzed. Similarly, because characters are assumed to be produced by the VA38 character generator, u_2 was constrained to be 1. Even more design parameters were found to be fixed when the possible implementations were enumerated. u_1 was found to be 1, indicating that the computer could not find characters to be plotted as rapidly as the display processor could plot them.

Because $u_1=1$, $u_5=0$ in accordance with Equation 5.1. Furthermore, Equation 4.53 states that $u_7=0 \Longrightarrow u_1=0$. The contrapositive of this implication is $u_1=1 \Longrightarrow u_7=1$. Consequently, $u_7=1$ for this application.

5.3.2 A Graph Theory Program

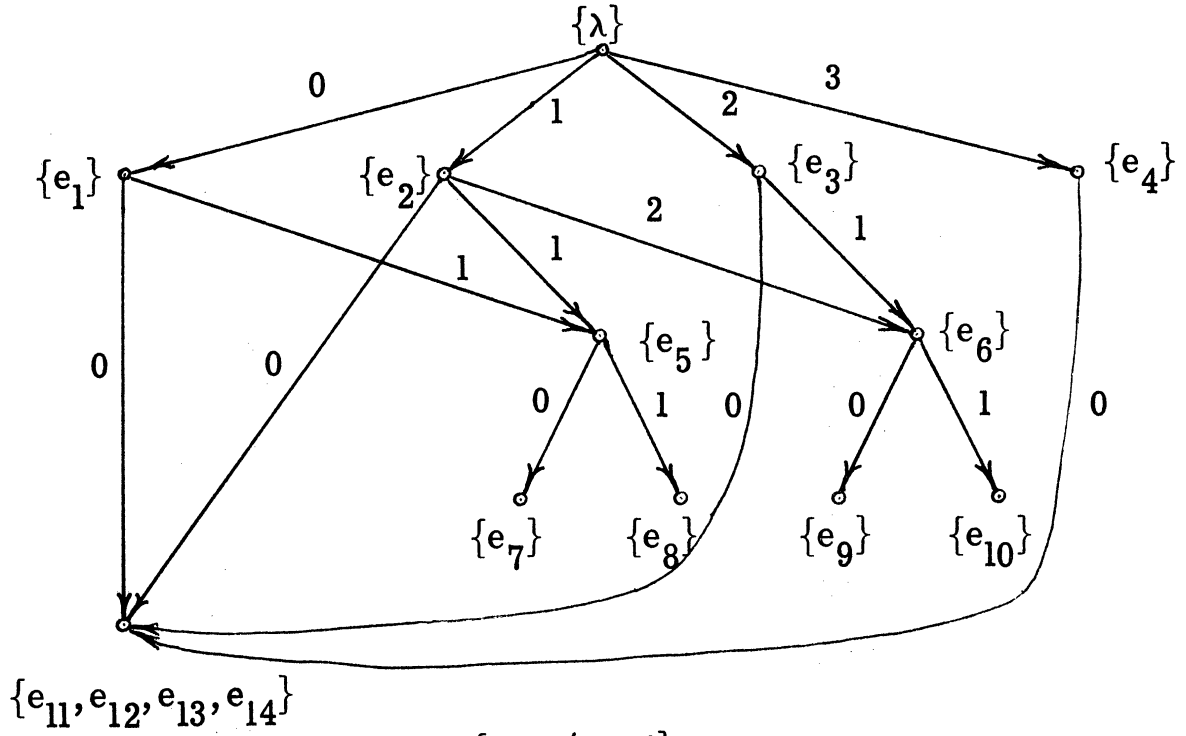
The second example to be considered is a program which allows the user to draw and modify undirected graphs. The forms which the topological structure may assume for a specific graph are shown in Figure 5.5. The graph which these structures represent consists of the four vertices e_1, e_2, e_3, and e_4 and the two edges e_5 and e_6. Each edge is considered to be a part of each vertex which it connects. In this example, e_5 connects e_1 to e_2, and e_6 connects e_2 to e_3. In addition to the edges which connect each vertex to other vertices, each vertex also has as one of its parts a vertex symbol. These vertex symbols are the entities e_{11}, e_{12}, e_{13}, and e_{14}. Each edge is composed of two line segments, so that it may be easily bent. These line segments are the entities e_7, e_8, e_9, and e_{10} in the figure. The entities e_{15}, e_{16}, e_{17}, and e_{18} in Figure 5.5b are artificial entities whose purpose is to position the vertices.

The sequence $\gamma(\rho(e,e'))$ in the structure shown in Figure 5.5b which are not $(I, G(D[e']), I)$ are the following:

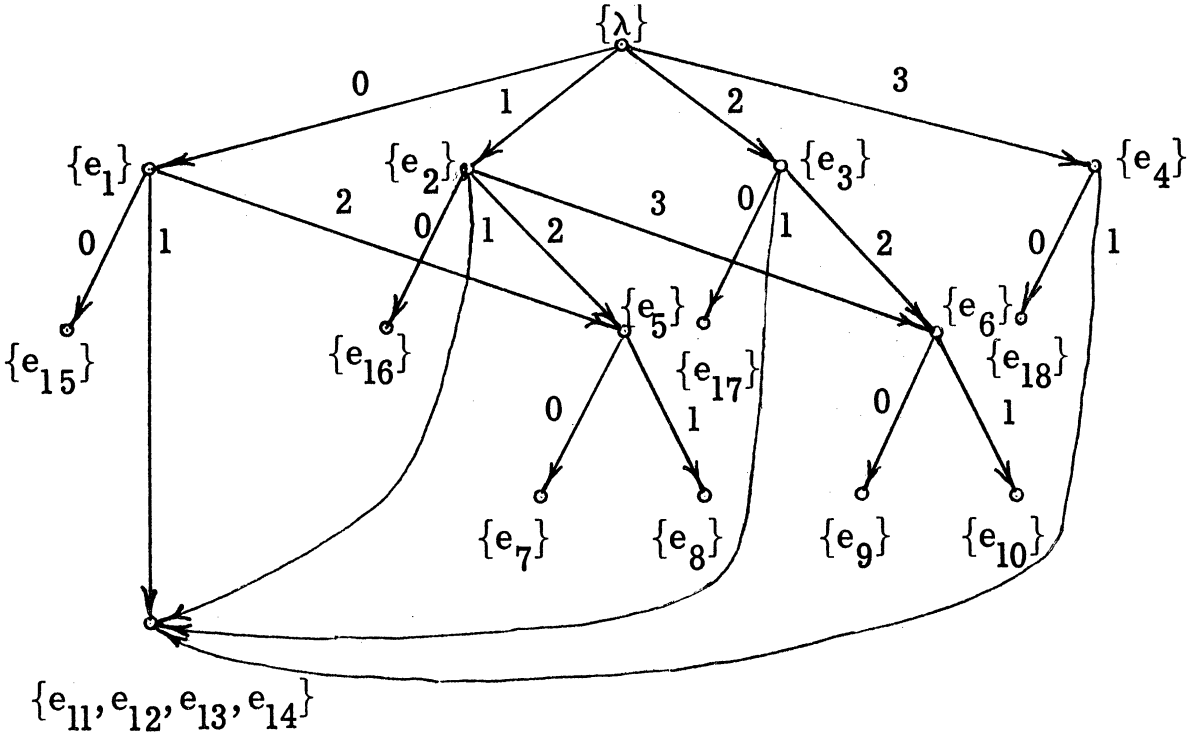
$$\gamma(\rho(\lambda, e_i)) = (I, G(D[e_i]), G(D[e_i])^{-1}),\ i = 1,2,3,4$$

$$\gamma(\rho(e_1, e_5)) = (I, G(D[e_5]), G(D[e_5])^{-1})$$

$$\gamma(\rho(e_2, e_6)) = (I, G(D[e_6]), G(D[e_6])^{-1})$$



a. $\{B, E/D, F'\}$



b. $\{B, E/D, \mu, \gamma\}$

Figure 5.5

Topological Structures for Graph Theory

$$\gamma(\rho(e_2, e_5)) = (G(D[\, e_5]\,)^{-1}, G(D[\, e_5]\,), I)$$

$$\gamma(\rho(e_3, e_6)) = (G(D[\, e_6]\,)^{-1}, G(D[\, e_6]\,), I)$$

Furthermore, $G(\{e_{11}, e_{12}, e_{13}, e_{14}\}) = I$. With these values, each vertex may be moved independently by modifying e_{15}, e_{16}, e_{17}, and e_{18}, and the head and tail systems of each edge are the tail systems of the vertices which it connects.

5.3.2.1 Estimation of Parameters

The parameters which describe the graph theory program will be expressed as functions of n_v, the number of vertices in the graph, n_e, the number of edges in the graph, and ℓ, the average number of points in each line segment which constitutes an edge. The control language inputs which will be considered invoke the following responses:

k_1: Create a vertex.

k_2: Destroy a vertex and all edges connected to it.

k_3: Create an edge between two specified vertices.

k_4: Destroy an edge.

k_5: Move vertex and adjust all edges connected to it so that other vertices are not moved.

k_6: Bend an edge.

Tables 5.7 through 5.9 show values of parameters for the graph theory application. When these values were estimated, each vertex symbol was assumed to be a cross (+) which was drawn with three invisible vectors and two visible vectors. (The length of each arm of this cross was assumed to be 32 points.) Although the assumed formats for subpictures for classes in E_0/D_0 shown in Table 5.1 require the same amount of storage for either value of u_2, $S_{\beta f}$ differs from $S_{\beta f}'$ for this application. The reason for this difference is that an additional vector must be included in the subpicture for each line segment which is part of an edge whenever $u_2 = 1$ so that the display processor coordinates are restored before Step 4 of Algorithm 3.1 is performed.

The parameters G_{mg}, G_{mg}', G_{mg}'', G_{hg}, G_{hg}', and G_{hg}'' which are shown in Table 5.9 are essentially the average times required to perform Algorithm 5.1 for various values of u_1, u_3, and u_7. The elements of $B \cup E/D$ which are examined by this algorithm are easily determined from Figure 5.5b and the above discussion of values of γ. From the discussion, an inverse of a value of G is seen to be computed once for each vertex and twice for each edge. Whenever $G(D[e])^{-1}$ is computed for a vertex e, the set of elements of B which are examined is $\rho(\{e\} \times S(e))$. Values of G' and G'' are equal only for the first two elements of this set. Consequently, the elements

Parameter	Value
q_{bf}	$2n_v + 4n_e$
q_{bg}	$3n_v + 4n_e$
q_{df}	$n_v + 3n_e + 2$
q_{dg}	$2n_v + 3n_e + 2$
q_{d0f}	$2n_e + 1$
q_{d0g}	$n_v + 2n_e + 1$
q_{sf}	$2n_v + 6n_e$
q_{sg}	$3n_v + 6n_e$
q_{s0f}	$n_v + 4n_e$
q_{s0g}	$2n_v + 4n_e$

Table 5.7

Structure Size Parameters for Graph Theory Program

Parameter	Value (locations)
s_0	2
S_t	1/9
S_t'	1/9
S_p	5/6
S_s	5/3
S_j	5/3
S_a	5/3
S_r	2/3
S_h	2/3
S_f	2
S_f'	$(6n_v + 12n_e)/q_{bf}$

Table 5.8

Storage Parameters for Graph Theory Program

Parameter	Value (locations)
$S_{\beta f}$	$(6n_e + 11)/q_{d0f}$
$S_{\beta f}'$	$(10n_e + 11)/q_{d0f}$
S_g	2
S_g'	4
S_γ	1/9
S_γ'	$4 - (2n_v + 4n_e)/q_{bg}$
$S_{\beta g}$	$(6n_e + 3n_v + 11)/q_{d0g}$
$S_{\beta g}'$	$(6n_e + 3n_v + 11)/q_{d0g}$

Table 5.8 (cont.)

Storage Parameters for Graph Theory Program

Parameter	Value (milliseconds)	Instruction Sequences
T_p	.005	1
T_j	.002	
T_s	.004	
T_j'	.007	2
T_s'	.007	2
T_b	.008	3
T_c	.005	4
T_{tf}	$.0024\ell n_e + .1536 n_v$	
T_{df}	$.0056\ell n_e + .1696 n_v$	
T_{tg}	$.0012\ell n_e + .0768 n_v$	
T_{dg}	$.0044\ell n_e + .2464 n_v$	
G_s	.01	5, 6
G_s'	.003	

Table 5.9

Time Parameters for Graph Theory Program

Parameter	Value (milliseconds)	Instruction Sequences
G_s''	.02	5, 6
G_s'''	Not estimated	
G_{mf}	.016	7
G_{mf}'	$(.003n_v+.009n_e)/g_{bf}$	
G_{hf}	.022	8
G_{hf}'	$(.003n_v+.009n_e)/q_{bf}$	
G_{mg}	$.184n_e/q_{sg}$	17
G_{mg}'	$.033n_e/q_{sg}$	
G_{mg}''	$.1915n_e/q_{sg}$	17
G_{hg}	$(.224n_e+.178n_v)/q_{sg}$	
G_{hg}'	$(.037n_e+.033n_v)/q_{sg}$	
G_{hg}''	$(.224n_e+.178n_v)/q_{sg}$	
G_{bf}	$(.012q_{df}-.004q_{d0f})/q_{sf}$	

Table 5.9 (cont.)

Time Parameters For Graph Theory Program

Parameter	Value (milliseconds)	Instruction Sequences
G_{bg}	$(.012q_{dg}-.004q_{d0g})/q_{sg}$	
G_{cf}	$(.164n_e+.177n_v)/q_{s0f}$	18
G_{tf}	$(.028n_e+.015n_v)/q_{s0f}$	
G_{tf}'	$(.02n_e+.011n_v)/q_{s0f}$	
G_{cg}	$(.164n_e+.341n_v)/q_{s0g}$	18
G_{tg}	$(.028n_e+.022n_v)/q_{s0g}$	
G_{tg}'	$(.02n_e+.016n_v)/q_{s0g}$	
$\bar{r}_g$	.15	
r_c	.006	9
$\bar{r}_d$	.15	
R_p	.005	10
R_j	.046	11
R_f	.012	12
R_f'	$.012+.082(n_v+2n_e)/q_{bf}$	16

Table 5.9 (cont.)

Time Parameters for Graph Theory Program

Parameter	Value (milliseconds)	Instruction Sequences
R_g	.012	12
R_g'	.082	13
R_γ	.007	14
R_γ'	$.012+(.036n_e+.018n_v)/q_{bg}$	15
R_{sf}	$.0395q_{bf}+.02q_{df}-.005q_{d0f}+.00325n_e-.11525$	
R_{sf}'	$.0475q_{bf}+.02q_{df}-.005q_{d0f}+.00975n_e-.13775$	
R_{sf}''	$.0435q_{bf}+.022q_{df}-.007q_{d0f}+.00325n_e-.12525$	
R_{sg}	$.0395q_{bg}+.02q_{dg}-.005q_{d0g}+.00325n_e-.11525$	
R_{sg}'	$.0475q_{bg}+.02q_{dg}-.005q_{d0g}+.00975n_e-.13775$	
R_{sg}''	$.0435q_{bg}+.022q_{dg}-.007q_{d0g}+.00325n_e-.12525$	

Table 5.9 (cont.)

Time Parameters for Graph Theory Program

$\{B, E/D, F'\}$

	k_1	k_2	k_3	k_4	k_5	k_6
ℓ_1	1	0	3	0	$6n_e/n_v$	3
ℓ_2	0	$1+6n_e/n_v$	0	3	$6n_e/n_v$	3
ℓ_3	2	0	4	0	$1.5+9n_e/n_v$	3
ℓ_4	0	$2+8n_e/n_v$	0	4	$1.5+9n_e/n_v$	3
ℓ_5	0	$2n_e/n_v$	0	1	$1.5n_e/n_v$	4.5
ℓ_6	0	$2n_e/n_v$	0	1	$3n_e/n_v$	1.5
ℓ_7	0	0	0	0	0	0
ℓ_8	0	0	0	0	0	0
ℓ_9	0	0	0	0	0	0
ℓ_{10}	0	$1+4n_e/n_v$	0	1	$1.5+6n_e/n_v$	1.5
ℓ_{11}	0	0	0	0	0	3
ℓ_{12}	0	$6n_e/n_v$	0	2	$9n_e/n_v$	3

Table 5.10

Values of the Function $\bar{n}$ for Graph Theory Program

	k_1	k_2	k_3	k_4	k_5	k_6
ℓ_{13}	0	0	0	0	$1.5+7.5n_e/n_v$	3
ℓ_{14}	0	0	0	0	0	0
ℓ_{15}	0	0	0	0	0	0

$\{B, E/D, \mu, \gamma\}$

	k_1	k_2	k_3	k_4	k_5	k_6
ℓ_1	2	0	3	0	$1.5+6n_e/n_v$	3
ℓ_2	0	$2+6n_e/n_v$	0	3	$1.5+6n_e/n_v$	3
ℓ_3	3	0	4	0	0	0
ℓ_4	0	$3+8n_e/n_v$	0	4	0	0
ℓ_5	0	$2n_e/n_v$	0	1	0	1.5
ℓ_6	0	$1+2n_e/n_v$	0	1	$1.5+3n_e/n_v$	4.5
ℓ_7	0	0	0	0	0	0
ℓ_8	0	0	0	0	$1.5+6n_e/n_v$	3

Table 5.10 (cont.)

Values of the Function $\bar{n}$ for Graph Theory Program

	k_1	k_2	k_3	k_4	k_5	k_6
ℓ_9	0	0	0	0	0	0
ℓ_{10}	0	$1+4n_e/n_v$	0	1	$1.5+6n_e/n_v$	1.5
ℓ_{11}	0	0	0	0	0	3
ℓ_{12}	0	$1+6n_e/n_v$	0	2	$1.5+9n_e/n_v$	6
ℓ_{13}	0	0	0	0	0	0
ℓ_{14}	0	0	0	0	$3n_e/n_v$	0
ℓ_{15}	0	0	0	0	$1.5+6n_e/n_v$	3

Table 5.10 (cont.)

Values of the Function $\bar{n}$ for Graph Theory Program

of E_0/D_0 which are examined are the class which contains the artificial entity in S(e) and the class which contains the vertex symbols. The only element of $E/D-E_0/D_0$ which is examined is D[e] . Whenever $G(D[\ e'])^{-1}$ is computed for an edge e', the two elements of $\rho(\{e'\} \times S(e'))$ are examined. For each element $b \epsilon \rho(\{e'\} \times S(e))$, $G'(b) = G''(b)$. Consequently, the elements of E_0/D_0 which are examined are the two classes which contain line segments which constitute e'. The only element of $E/D-E_0/D_0$ which is examined is D[e'] .

In order to estimate the parameters R_{sf}, R_{sf}', R_{sf}'', R_{sg}, R_{sg}', and R_{sg}'', the elements of $B \cup E/D$ which are examined by Algorithm 3.5 must also be identified. However, in order to estimate these parameters accurately, the elements of E/D to which Operation ℓ_5 is applied for this application must first be identified. These elements of E/D can be identified only after the basic operations which constitute the response to each control language input have been considered.

The average number of times that each basic operation is executed in order to respond to each of the inputs k_1, k_2, k_3, k_4, k_5 and k_6 is shown in Table 5.10. As in the editor example, these inputs were assumed to be equally likely in order to compute values of f_f and f_g. The values shown in the table for inputs k_5 and k_6 are actually 1.5 times the values needed to move a vertex once or to bend an edge once, for these inputs are assumed to be terminations of continuous user

activity. For example, in order to move a vertex, the user might aim the light pen at the vertex and then move the pen. The vertex is then positioned many times at the coordinates of the light pen to give the appearance of continuous movement. When the user terminates his action by removing the light pen, an average of half of the process of moving the vertex to the last light pen coordinates sampled has not been completed. Since the user generally has moved the light pen since these coordinates were sampled, the position of the vertex must be updated once more to insure that its final position is correct. Then, since the input k_5 is considered to be the termination of user activity, the average number of times that each basic operation must be performed in order to respond to this input is 1.5 times the number of times that it must be performed to move the vertex once. (Although movement of the vertex while the user is inputting coordinates with the light pen might also be considered to be the response to an input, the time required to perform this response is generally not critical if the pen can be tracked at a sufficient speed.)

From Table 5.10, Operation ℓ_5 must be performed in order to respond to the inputs k_2, k_4, k_5, and k_6 for the structure $\{B, E/D, F'\}$, and in order to respond to the inputs k_2, k_4, and k_6 for the structure $\{B, E/D, \mu, \gamma\}$. With the exception of 2/3 of the time that this operation is performed in order to respond to k_6 for the structure $\{B, E/D, F'\}$, this operation is executed in order to identify the elements of $P(D[e])$

such that e represents an edge. Whenever Operation ℓ_5 is applied in this way and is performed by Algorithm 3.5, all but two elements of B, those in $\rho(\{e\}\times S(e))$, are examined. Similarly, all elements of E/D, other than the two classes which contain line segments which constitute e, are examined. However, 2/3 of the time that Operation ℓ_5 is executed in order to respond to k_6 for the structure $\{B,E/D,F'\}$, it is applied to identify the elements of a set P(D[e']), where e' is a vertex. Whenever Operation ℓ_5 is applied in this way and is performed by Algorithm 3.5, all of the elements of $B-\rho(\{e'\}\times S(e'))$ are examined, and all elements E/D are examined. Then, by considering the probability of each input, the average times R_{sf}, R_{sf}', R_{sf}'', R_{sg}, R_{sg}', and R_{sg}'' which are required to perform Algorithm 3.5 for various values of u_1, u_3, and u_7 can be estimated. However, if B is assumed to be large when compared to the average number of edges per vertex, the error introduced by assuming that ℓ_5 is always applied to identify the elements of P(D[e]) such that e is an edge is not significant. This assumption was made when the values of these parameters shown in Table 5.9 were estimated.

5.3.2.2 Results of Analysis

The programs which are described in Appendix B were applied to the parameters and operation frequences which were estimated above in order to compare various implementations of the structure for various values of n_v, n_e, and ℓ. For a small number of edges (about 1/4 the number of vertices), 208 implementations satisfied

the constraint $g < t_f$. However, for larger numbers of edges, 224 implementations satisfied this constraint.

The costs of eight selected implementations which satisfied this constraint for all values of n_v and n_e tested are plotted in Figures 5.6 through 5.8. Whenever s_{max} and t_{max} are sufficiently large (i.e., whenever $s_{max} \geq 11920$ locations and $t_{max} \geq 122$ msec), the implementation 01001011010 is the optimum implementation for all values of n_v and n_e tested. However, if s_{max} is reduced to 8959 locations, the optimum implementation becomes 01001010000. Note that this implementation differs from 01001011010 only in that references to values of μ are not stored and the ring which represents each set P(D[e]) is not stored. If either $s_{max} < 8959$ locations or $t_{max} < 122$ msec, a structure which describes a graph with $n_v = 140$, $n_e = 175$, and $\ell = 100$ cannot be implemented. For all values of s_{max} such that $8959 < s_{max} < 11920$, the cost of the optimum implementation is greater than the cost of the implementation 01001011010 and less than the cost of the implementation 01001010000.

However, the enumeration of the implementations for which $u_1 = 0$ showed that g was only slightly less than t_f for these implementations. Since t_f is the total time required to plot entities in E_0 whenever $u_1 = 0$, the fact that g is only slightly less than t_f indicates that the optimum implementations discussed above may not be practical. In

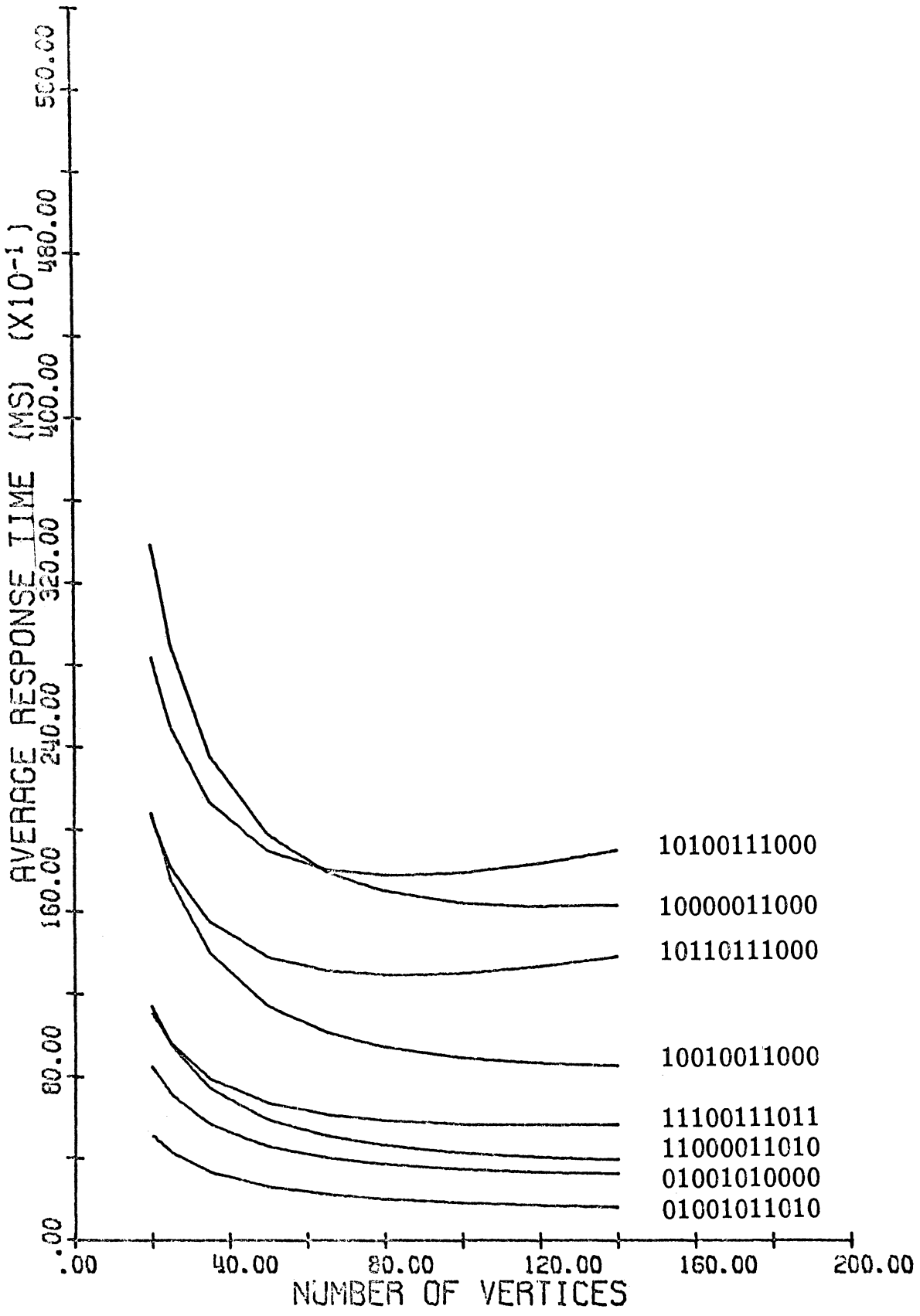


Figure 5.6

Graph Theory Program Cost for $n_e = 100$ and $\ell = 100$

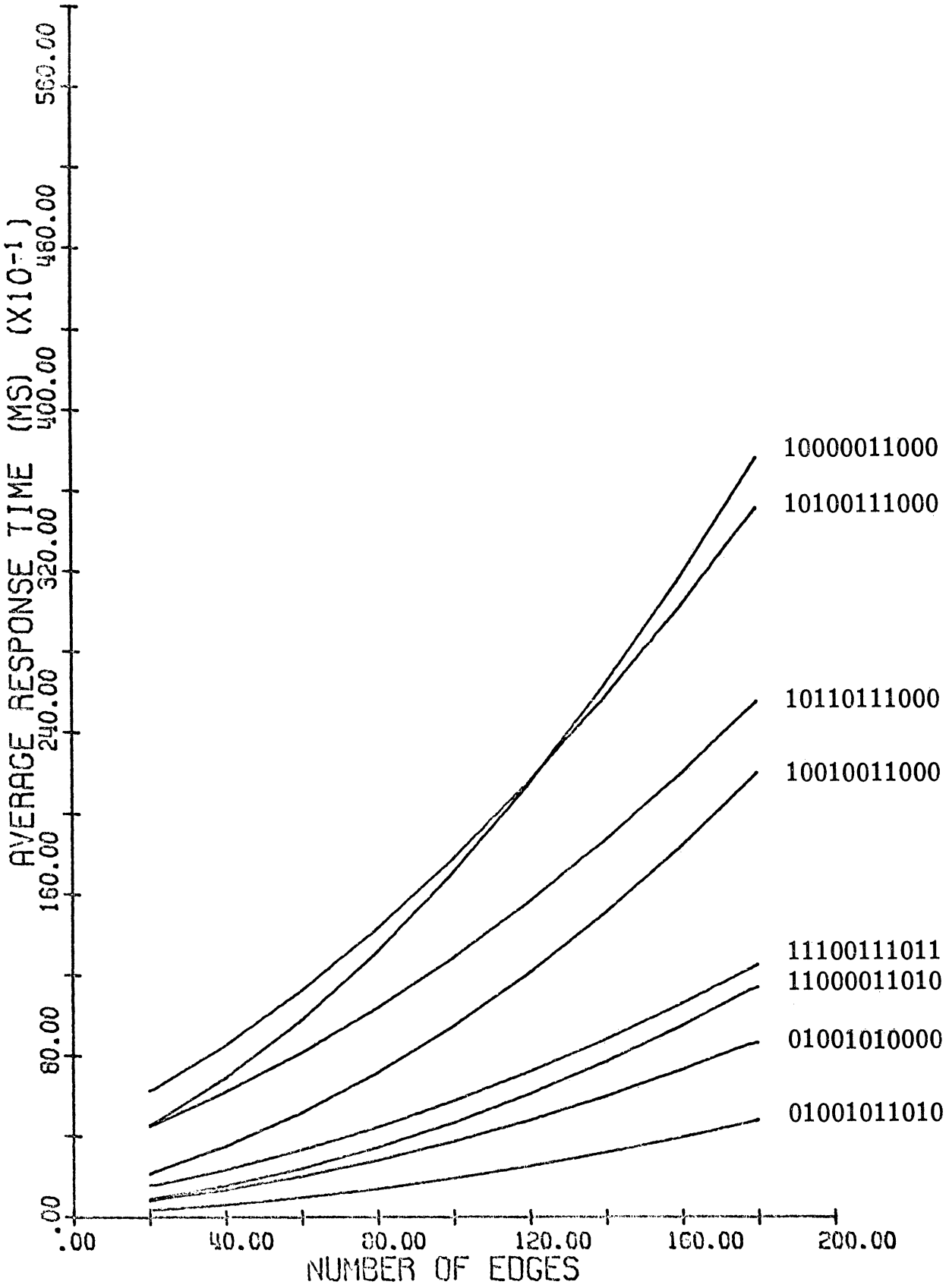


Figure 5.7

Graph Theory Program Cost for $n_v = 80$ and $\ell = 100$

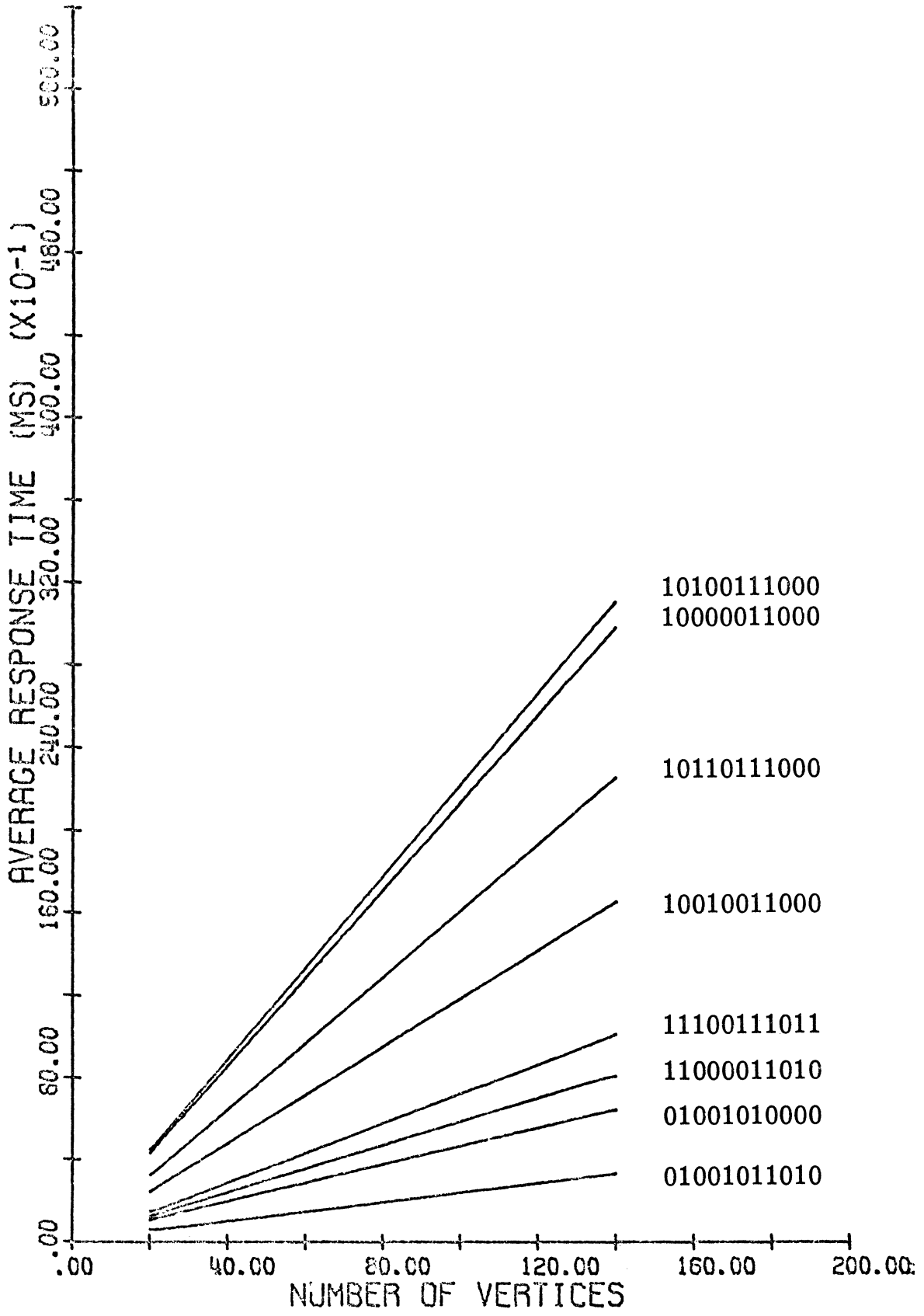


Figure 5.8

Graph Theory Program Cost for $n_e/n_v = 5/4$ and $\ell = 100$

particular, in the course of drawing a graph for which $\ell = 100$ points, the user may produce an intermediate graph for which $\ell < 100$ points. If $u_1 = 0$, t_f may then approach g for this intermediate graph, and thus prevent response to further user inputs. To test this possibility, the implementations for $n_v = 80$, $n_e = 100$, and $\ell = 50$ were enumerated. Only 128 implementations, all with $u_1 = 1$, were found to satisfy the constraint $g < t_f$.

Because implementations with $u_1 = 0$ may not be practical for this application, an attempt was made to find the optimum implementation for which $u_1 = 1$ (assuming that s_{max} and t_{max} were very large). However, no optimum was found for all values of n_e and n_v tested. For $n_v \leq 25$ and $n_e = 100$, the optimum implementation was found to be 11100111011, whereas, for other values of n_v and n_e tested, the optimum implementation was found to be 11000011010. The significant difference between these two implementations is that one is an implementation of the structure $\{B, E/D, \mu, \gamma\}$, whereas the other is an implementation of the structure $\{B, E/D, F'\}$. The structure $\{B, E/D, F'\}$ is better when n_e/n_v is small because $t_f/(t_f-g)$ is smaller for this structure. However, when n_e/n_v is large, the sum of the frequencies of the operations which must synchronize with the picture generator is larger for the structure $\{B, E/D, F'\}$ than for the structure $\{B, E/D, \mu, \gamma\}$. In particular, $f_f(\ell_4) > f_g(\ell_3) + f_g(\ell_4) + f_g(\ell_{15})$

for $n_e/n_v > 8/3$. Consequently, $\{B, E/D, \mu, \gamma\}$ is the better structure for large n_e/n_v.

Other implementations whose costs are plotted in Figures 5.6 through 5.8 were found to be the best implementations with further constraints on the design decisions. The implementations 10110111000 and 10010011000 were found to be the best implementations for $u_1 = 1$ and $u_2 = u_{10} = 0$ (assuming large s_{max} and t_{max}). Since $u_{10} = 0$, these implementations use Algorithm 3.5 to perform Operation ℓ_5. If u_4 is also constrained to be zero (i.e., Algorithm 3.5 is not used to perform Operation ℓ_5), the best implementations are 10100111000 and 10000011000. Note that again no optimum implementation exists for all values of n_e and n_v tested. The worst possible implementations for this application are 10100010100 and 10000010100. The costs of these implementations are insignificantly different than the costs of the implementations 10100111000 and 10000011000 shown in Figures 5.6 through 5.8.

5.4 Conclusion

The application of the cost function to two applications for a specific hardware configuration has been demonstrated. From the results obtained for these applications, implementations with $u_1=0$ have been shown to often not satisfy the constraint $g<t_f$. Consequently, the picture generation process must often be interrupted in order to

respond to user inputs if one of these implementations is used. The nonexistence of a unique optimum has also been demonstrated. However, even when a unique optimum does not exist, the cost function may be applied to select the implementations which should be considered.

Chapter 6

CONCLUSIONS

6.1 General Design Principles

Three major design principles have been illustrated in this paper. The first of these is that the accepted method of representing coordinate transformations, namely as values of the function F', is not necessarily the best method for all applications. In Chapter 2, the structure $\{B, E/D, \mu, \gamma\}$, in which coordinate transformations were represented differently, was described. Through the application of the cost function which was developed in Chapters 3 and 4, this structure was shown in Chapter 5 to provide faster response than the conventional structure $\{B, E/D, F'\}$ for certain applications. The structure $\{B, E/D, \mu, \gamma\}$, however, is not the only other structure which can be specified. For example, a hybrid of these two structures might be developed so that concatenation constraints are enforced as a consequence of refreshing the picture, but no artificial entities are needed in order to manipulate some entities independently of other entities.

Secondly, the danger of using a computer program as the picture generator whenever the picture generation process is to have priority over the response to user inputs has been illustrated. In this paper, the picture generation process was assumed to have this priority, and the constraint $g < t_f$ was imposed to insure a finite response time.

As illustrated by the graph theory example in Chapter 5, whether or not a computer program may be used as the picture generator may even depend on the behavior of the user. For this example, the optimum picture generator is a computer program if the user consistently draws edges of sufficient length throughout his session with the computer. However, if he draws edges of greatly varying lengths, the computer will cease to respond if the average of these lengths becomes too small.

Thirdly, the necessity to synchronize a program which is modifying the structure with the picture generator in order to perform certain operations on the structure was illustrated. By always waiting for the picture generator to reach a particular part of the structure, rather than interrupting the picture generator and restarting it with the first step of Algorithm 3.1, a program which operates on the structure does not interfere with the picture generation process. Although an operation on the structure is generally completed more rapidly if it is performed while the picture is not being generated, the picture may momentarily disappear if many operations which modify the structure are performed in this way in rapid succession. Consequently, responses such as continuous movement or deformation of entities are not practical unless the program which is modifying the structure is synchronized with the picture generator.

6.2 Limitations

A significant consideration which has been ignored throughout this paper is that a topological structure may not contain suitable information for generating two-dimensional projections of three-dimensional pictures. In particular, if hidden lines or surfaces are not to be displayed, the position and geometry of each entity in E_0 must be known before any entity in E_0 can be displayed. Consequently, in order to display this type of picture, a geometric description of the picture must first be formed from the topological structure. However, for some applications, a suitable projection is produced if only the lines or surfaces in each entity in E_0 which are obstructed from view by other parts of that entity are not displayed. This projection may easily be generated from the topological structure by Algorithm 3.1 with $u_2 = 0$.

6.3 Strengths and Weaknesses

The development in this paper may be applied to an existing hardware configuration to determine the best way to implement a topological structure for a proposed application. This ability is useful to the programmer of computer display equipment, for he is often given the task of utilizing an existing hardware configuration, but he has no control over the design of that configuration. Because some operations on the structure require synchronization with the

picture generator, the topological structure is expected to be supported by a system program, rather than by each individual application program. However, at the time that such a system program is written, the major applications of the topological structure generally are known. The discussion in this paper may then be applied to these major applications to determine which implementations of the structure should be considered.

The estimation of parameters for a particular application, however, is a somewhat tedious process. In the worst case, 106 of these parameters are not zero, and, therefore, must be estimated. However, many of these parameters (e.g., T_p, T_j, T_s, S_p, etc.) assume the same value for all applications of a particular hardware configuration, and need not be estimated separately for each application.

Appendix A

AN IMPLEMENTATION OF THE STRUCTURE $\{B, E/D, \mu, \gamma\}$

The structure $\{B, E/D, \mu, \gamma\}$ which was described in Chapter 2 has not been implemented previously. For this reason, the author has developed a program for a DEC 339 [1, 52] to demonstrate the feasibility of implementing this structure, as well as to provide a tool for experimenting with several applications of the structure. This program includes a compiler for a graphically-inputted language which is similar in form to that described by W. M. Newman [45,46] . This compiler allows the user to specify an application program by drawing a state diagram of it and describing each state. The compiler is designed to facilitate preparation of programs which respond to user inputs by modifying and interpreting the structure, but it is poorly suited to the preparation of other types of programs.

The implementation of the structure for this program is described below. The graphically-inputted state diagram language is then described.

A.1 Implementation of the Structure

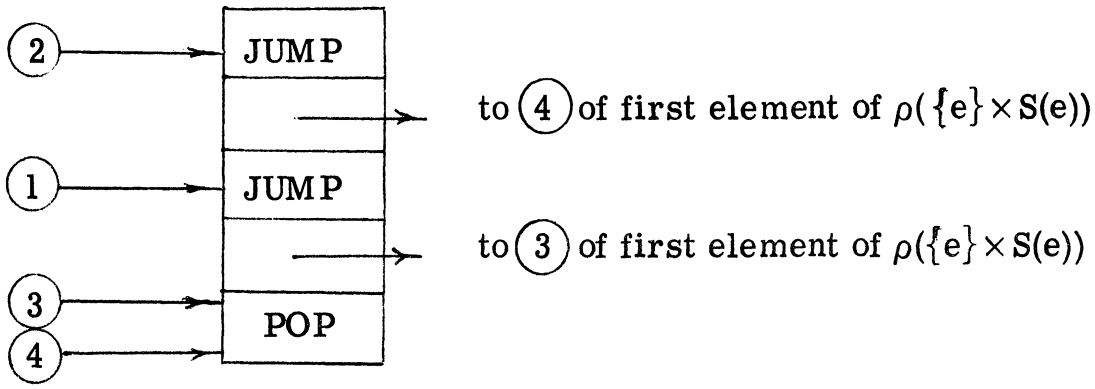
The hardware configuration for which this program was implemented does not include any form of back-up storage (e.g., magnetic tape, disk, etc.). Consequently, the entire program is resident in core memory whenever any part of it is being executed. For this reason, only a small amount of storage is available for the structure, and the design

parameters for the structure were chosen such that redundant information is not stored. However, although storage is used ineffectively if $u_1 = 1$ (i.e., the display processor can interpret only the low order 12 bits of each 18-bit word), u_1 was chosen to be 1 so that the execution of Algorithms 3.1 and 5.1 by the display processor could be demonstrated. Furthermore, u_4 was chosen to be zero so that the use of the picture generator to perform Operation ℓ_5 could be demonstrated. The values chosen for the design parameters were then $u_1 = u_2 = u_3 = u_7 = 1$ and $u_4 = u_5 = u_6 = u_8 = u_9 = u_{10} = u_{11} = 0$.

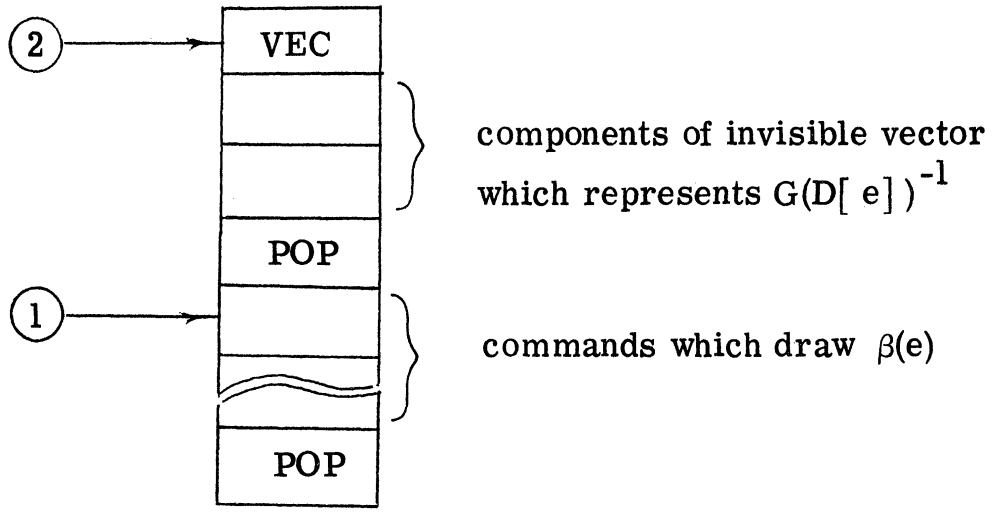
The formats which were chosen for the storage blocks which represent elements of $B \cup E/D$ are shown in Figures A.1 and A.2. The display processor commands which appear in these formats perform the following functions:

JUMP: Jump command (2 words). Transfer control of the display processor to the location whose address appears in the second word of this command.

PJMP: Push jump command (2 words). Call a display processor subroutine by first saving the address of the next command on a push-down stack in the PDP-9's memory, and then transferring control of the display processor to the location whose address appears in the second word of the PJMP command.



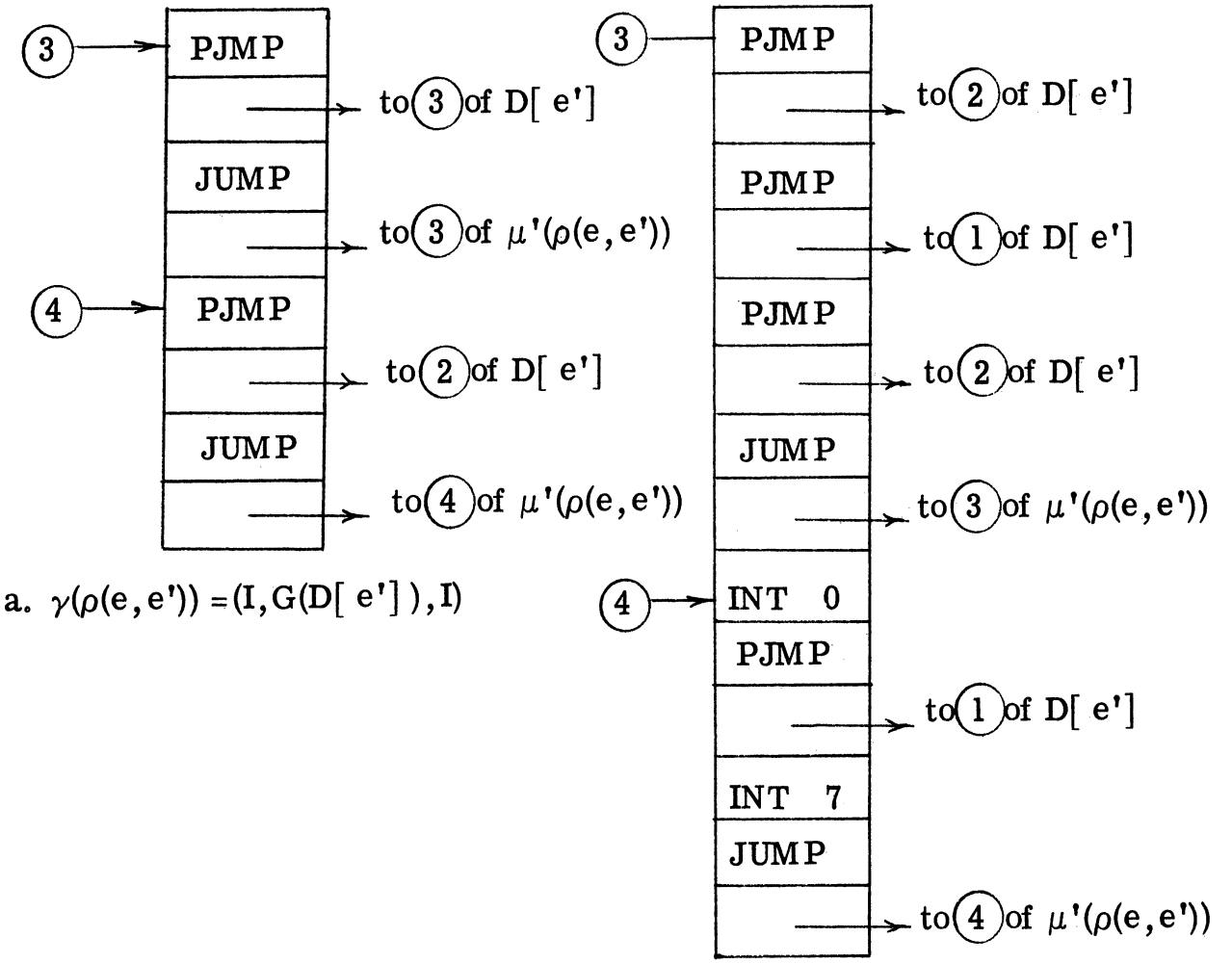
a. Block Format for $D[\ e] \in E/D - E_0/D_0$



b. Block Format for $D[\ e] \in E_0/D_0$

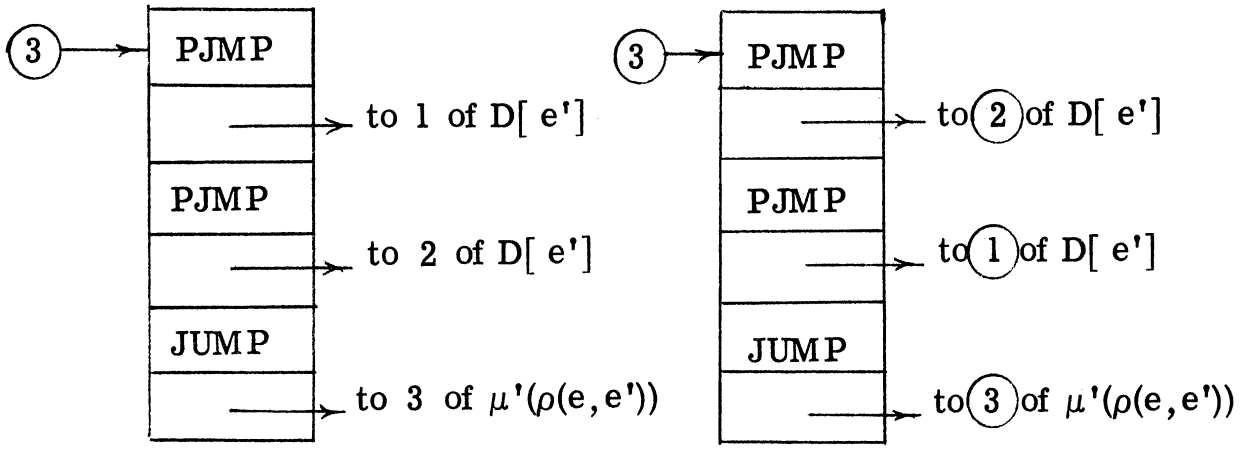
Figure A.1

Block Formats for Elements of E/D



a. $\gamma(\rho(e,e')) = (I, G(D[\,e'\,]), I)$

b. $\gamma(\rho(e,e')) = (G(D[\,e'\,])^{-1}, G(D[\,e'\,]), G(D[\,e'\,]$



c. $\gamma(\rho(e,e')) = (I, G(D[\,e'\,]), G(D[\,e'\,])^{-1})$

d. $\gamma(\rho(e,e')) = (G(D[\,e'\,])^{-1}, G(D[\,e'\,]))$

Figure A. 2

Block Formats for Elements $\rho(e,e') \in B$

POP: Pop command (1 word). Return from a display processor subroutine by first removing the last address from a push-down stack in the PDP-9's memory and then transferring control of the display processor to the location specified by this address.

INT n: Set intensity to level n (1 word). As used here, INT 0 renders all further display invisible until an INT 7 command is executed.

VEC: Enter vector mode (1 word). As used here, display the (invisible) vector whose components are stored in the next two locations, and then proceed with the command which follows the second of these components.

The type of element which each block represents is not stored explicitly in that block, for it may be easily determined by identifying the command which is stored in the first location in the block. If this location contains a VEC command, the block represents an element of E_0/D_0; if it contains a JUMP command, the block represents an element of $E/D-E_0/D_0$; if it contains a PJMP command, the block represents an element of B.

Each block has one or more "entry points", i.e., locations at which the display processor may begin execution of that block. The various entry points are labeled (1), (2), (3), and (4) in Figures

A.1 and A.2 and have the following significance:

(1): The display processor begins at this entry point of a block which represents an element $D[e] \in E/D$ in order to plot an entity in $D[e]$ after $H(e)$ has been computed.

(2): The display processor begins at this entry point of a block which represents an element $D[e] \in E/D$ in order to determine $G(D[e])^{-1}$.

(3): Immediately after finishing with the element $\rho(e,e') \in B$, the display processor begins at this entry point of the block which represents $\mu'(\rho(e,e')) \in B \cup E/D$ in order to continue plotting an entity in $D[e]$.

(4): Immediately after finishing with the element $\rho(e,e') \in B$, the display processor begins at this entry point of the block which represents $\mu'(\rho(e,e')) \in B \cup E/D$ in order to continue the process of determining $G(D[e])^{-1}$. For each $\rho(e,e') \in B$ for which no entry point (4) is shown, this entry point is considered to be the entry point (4) of the block which represents $\mu'(\rho(e,e'))$.

Each item in these storage formats has been considered to occupy an integer number of locations. Since the DEC 339 display processor can interpret only 12 bits of an 18-bit word, there are actually 6 bits

in each word into which other items could be packed. However, since each item in these formats occupies at least 12 bits, these items cannot be packed in accordance with the second restriction stated for Algorithm 4.1. Consequently, no generality is lost by assuming that each item occupies an integer number of locations.

The block whose format is shown in Figure A. 2b was not actually implemented, for it was thought not to be useful at the time that the program was written. Whereas a sequence $\gamma(\rho(e,e'))$ of the form $(G(D[\ e'])^{-1}, G(D[\ e]), G(D[\ e])^{-1})$ is much less frequently used than a sequence of any other form, its effect cannot be simulated through the use of artificial entities and values of γ which assume the other three forms. Consequently, provision for this sequence should be included in future programs which support the structure $\{B, E/D, \mu, \gamma\}$.

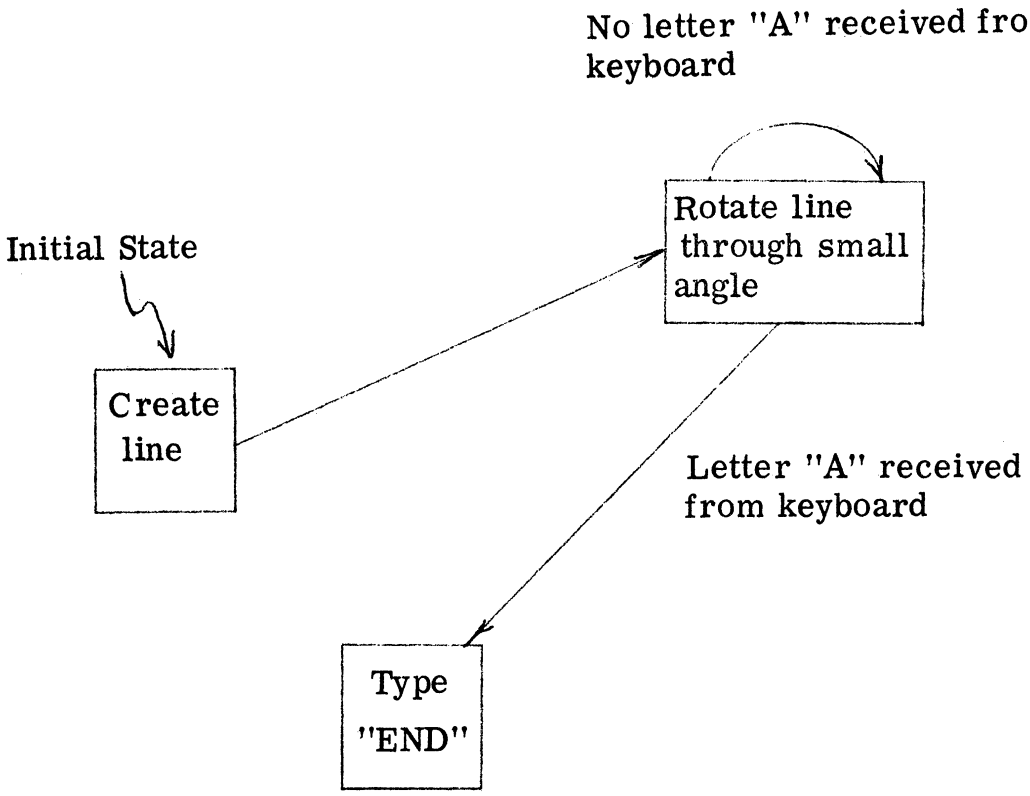
A. 2 The Display Language

As has been mentioned above, a compiler similar to that described by W. M. Newman [45,46] was included in the program. In order to specify an application program to this compiler, the user must specify both (1) a state diagram for the application program, and (2) the operations which are performed when the application program is in each state. These two parts of the language are described below.

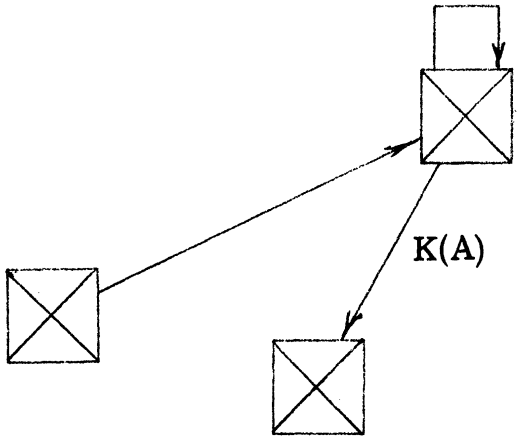
A. 2.1. The State Diagram

A state diagram for a program is essentially a flow chart in which inputs, as well as Boolean conditions, are specified to control the flow of execution. For example, Figure A.3a shows the state diagram for a simple program which creates a line and then continually rotates this line about one end point. When the letter "A" is received from the keyboard the message "END" is typed, and the program is terminated.

Figure A.3b shows this state diagram as it would be inputted to the compiler. Each box in the diagram represents a state. The "X" in each box in this diagram indicates that some operations are associated with that state. Each labeled path in the diagram represents a transition between two states which occurs upon completion of the operations associated with the initial state if the input designated by the label occurred while these operations were being executed. In this example, if the letter "A" is received from the keyboard (a condition designated by the label "K(A)") while the line segment is being rotated, a transition to the final state of the program occurs as soon as the line segment has been rotated to a new position. Each unlabeled path represents a transition between two states which occurs upon completion of the operations associated with the initial state if no other transition occurs. In this example, the unlabeled path from the initial state represents an unconditional transition, since there are no



a. A State Diagram



b. A State Diagram in Compiler Form

Figure A.3

Representation of State Diagrams

other paths from the initial state. The other unlabeled path, however, represents a transition which occurs only if the transition to the final state does not occur.

The information needed to form labels which are used to condition state transitions is shown in Table A.1. Each label consists of a device code followed by a character code in parentheses. (The loss-of-tracking device is an exception, since it is considered to have only one character.) Examples of labels are B(TEST), P(↑), L(1), L(53), K(*), R(A), and X. In the event that more than one of the inputs which condition transitions from a particular state is received while the program is in that state, only the one from the device which appears first in this table is interpreted when the operations associated with that state are completed.

The tracking pattern device (code P) indicated in this table is an interpretation of subsequent motion of the light pen after a light pen hit.* Whenever a light pen hit occurs on any entity and light pen tracking is not in progress, tracking is started and subsequent motion of the light pen is interpreted. The user may then produce a tracking pattern input by moving the light pen in one of the ways shown in Figure A.4. In this figure, the crosses denote positions of light pen hits,

*These interpretations of light pen motion were found to be useful in a previous program [25, 27]. A discussion of methods for interpreting light pen motion is given in a previous report by the author [27].

Device Code	Interpretation	Applicable Characters
B	Boolean input (test for nonzero)	All variable names (i.e., all combinations of from one to four alphanumeric characters, the first of which is alphabetic). Whenever the value of the designated variable is nonzero, the Boolean input is considered to have occurred.
X	Loss of tracking	No characters specified.
L	Light pen hit	Class numbers for E_0/D_0 (described in Section A. 2. 2 below).
P	Tracking Pattern	"*" for circular motion, "↑" for vertical stroking motion, "← " for moving away from the light pen hit.
K	Keyboard character	All ASCII characters
R	Paper tape reader character	All ASCII characters

Table A.1

Device Codes

a. Circular Motions (*)

b. Vertical Stroking Motions (↑)

c. Motions Away From Light Pen Hits (←)

Figure A.4

Example Tracking Pattern Characters

and the arrows denote paths of the pen. The tracking pattern characters are interlocked so that no more than one of them can occur between two successive losses of tracking.

A. 2. 2 Specification of Operations Associated With a State

The operations which are associated with each state are specified by a list of text statements. Each statement consists of a key word followed by a list of parameters. Each parameter is either a decimal integer between -999 and 999, a variable name, a text list, or an expression. With the exception of the special variable name *, each variable name is a string of from one to four alphanumeric characters, the first of which is alphabetic. The value of each variable is either a pointer to an element of $B \cup E/D$ or an integer. The special variable name * is preassigned the value $D[\lambda]$. A text list is a string of quoted phrases interspersed with variable names. For example, if the variable B1 has the value 5 and GLUT has the value -20, the statement

TYPE "THERE ARE " B1 " ITEMS AT " GLUT " DEGREES."

would cause the text

THERE ARE 5 ITEMS AT -20 DEGREES.

to be typed. (The quote character itself cannot be outputted.) An expression is used only in an ASSIGN statement and is a string of variable names and numbers separated by the binary operator symbols

"+", "-", "*", and "/", which represent addition, subtraction, multiplication, and division, respectively. In addition, the first symbol may be preceded by the unary operator symbol "-", which denotes unary minus. Any operator which appears to the left of a second operator has higher precedence than the second operator. For example, if X is a variable whose value is 6, the expression

X - X/2

has the value 0, rather than the value 3.

The function of each statement is shown in Table A. 2. In this table, p_i denotes the i th parameter in each statement. (All parameters which are not specified when these statements are written are assumed to be zeros.) To facilitate interpretation of light pen hits, the EXPOSE statement allows the application program to assign a class number (a number between 1 and 63) to each class in E_0/D_0. Whenever a light pen hit occurs on an entity in a class $D[e] \epsilon E_0/D_0$ which has been assigned the class number n, a light pen input (code L) with the character n is produced, and further light pen hits are disabled until the next UNBLOCK statement is executed.

As a simple example of the use of these text statements, consider the program whose state diagram is shown in Figure A. 3. In the initial state of this program, a vector with x and y components each of length 100 points may be created by the following statements:

Statement	Interpretation
ASSIGN p_1 expression	The expression is evaluated, and the resultant value is stored as the value of p_1.
BLANK p_1 p_2 p_3	A class in E_0/D_0 is created and a pointer to this class is stored as the value of p_3. The subpicture which is associated with this class is an invisible vector whose x component is p_2 and whose y component is p_1. (One of the ways of performing Operation ℓ_1.)
CHEW p_1	If the class in E/D which is referenced by p_1 is not currently part of the topological structure from which a picture is being generated, the storage occupied by this class is returned to the free storage pool. Furthermore, the storage occupied by all elements of $B \cup E/D$ which are needed to generate entities in this class, but which are not parts of the structure from which a picture is being generated, is returned to the free storage pool. (The initial step of this operation is Operation ℓ_2.)
DEFINE p_1 text list	A class in E_0/D_0 is created and a pointer to this class is stored as the value of p_1. The subpicture which is associated with this class is the string

Table A. 2

Text Statements

Statement	Interpretation
DEFINE (cont.)	of characters which is produced by scanning the text list. This subpicture is drawn from a point near the lower left hand corner of the first character to a point near the lower right hand corner of the last character. These points were chosen so that two of these subpictures appear as one text string when concatenated to each other. Each character is 8 points wide. (One of the ways of performing Operation ℓ_1.)
EXPOSE p_1 p_2	The class in E_0/D_0 which is referenced by p_1 is assigned the class number p_2.
FIND p_1 p_2 p_3	The x and y coordinates of the last light pen hit on an entity in a class which has the class number p_1 are stored as the values of p_2 and p_3, respectively. If p_1 is zero, the last x and y coordinates at which the tracking cross was displayed are stored as the values of p_2 and p_3, respectively.
GENERATOR p_1 p_2 p_3	A pointer to the class in E/D which is the second component of the p_1th element of B whose first component is referenced by p_2 is stored as the value of p_3. If there is no such class, p_3 is set to

Table A. 2 (cont.)

Text Statements

Statement	Interpretation
GENERATOR (cont.)	zero. (Corresponds roughly to Operation ℓ_6, followed by several Operations ℓ_{10}, followed by Operation ℓ_{12}.)
HOMOGENIZE p_1 p_2	A class in E_0/D_0 is created and a pointer to this class is stored as the value of p_2. The subpicture which is associated with this class is identical to that which is associated with the class which is referenced by p_1. (One of the ways of performing Operation ℓ_1.)
INSERT p_1 p_2 p_3 p_4 p_5	An element of B, whose first component is the class which is referenced by p_2 and whose second component is the class which is referenced by p_1, is created. The value of μ for the newly created element is the p_3th element of B whose second component is referenced by p_4. $\gamma(\rho(e,e'))$ for this new element $\rho(e,e') \epsilon B$ is $(G(D[e'])^{-1}, G(D[e']), I)$ if $p_5 < 0$, $(I, G(D[e']), I)$ if $p_5 = 0$, or $(I, G(D[e']), G(D[e'])^{-1})$ if $p_5 > 0$. (Corresponds roughly to Operation ℓ_3.)

Table A. 2 (cont.)

Text Statements

Statement	Interpretation
JUSTIFY p_1 p_2	The absolute value of p_2 is stored as the value of p_1.
LOCATE p_1 p_2 p_3 p_4	The x and y coordinates of the entity in the class which is referenced by p_2 the p_1th time that this class is encountered during the picture generation process are stored as the values of p_3 and p_4, respectively.
MEASURE p_1 p_2 p_3	The x and y coordinates which represent the value of G(D[e]) are stored as the values of p_2 and p_3, respectively, where p_1 references D[e] ϵ E/D. (Corresponds roughly to Operation ℓ_8.)
NONPRIMITIVE p_1	A block which is to represent an element of E/D-E_0/D_0 is created and a pointer to this block is stored as the value of p_1. (One of the ways of performing Operation ℓ_1.)
OWNER p_1 p_2 p_3	A pointer to the first component of the element of P(D[e]) which is on the push-down stack used by Algorithm 3.1 the p_1th time that D[e] is encountered

Table A. 2 (cont.)

Text Statements

Statement	Interpretation
OWNER (cont.)	during Step 2 of Algorithm 3.1 is stored as the value of p_3, where p_2 references the class D[e] . If this element of P(D[e]) does not exist, p_3 is set to zero. (Corresponds roughly to part of Operation ℓ_5.)
PUNCH text list	The result of scanning the text list is punched on paper tape.
QUERY p_1 p_2 p_3	A pointer to the second component of the p_2th element from the end of the push-down stack used by Algorithm 3.1 at the time of the last light pen input on an entity in a class in E_0/D_0 whose class number was p_1 is stored as the value of p_3.
REMOVE p_1 p_2 p_3	The p_2th element of B whose first component is referenced by p_1 and whose second component is referenced by p_3 is destroyed. (Corresponds roughly to Operation ℓ_4.)
SUBSTITUTE p_1 p_2 p_3 p_4	The second component of the p_2th element of B whose first component is referenced by p_1 and whose second component is referenced by p_3 is

Table A. 2 (cont.)

Text Statements

Statement	Interpretation
SUBSTITUTE (cont.)	replaced with the element of E/D which is referenced by p_4. (Corresponds roughly to Operation ℓ_{15}.)
TYPE text list	The result of scanning the text list is typed on the teletype.
UNBLOCK	Light pen hits are enabled.
VISIBLE p_1 p_2 p_3	A class in E_0/D_0 is created and a pointer to this class is stored as the value of p_3. The subpicture which is associated with this class is a visible vector whose x component is p_2 and whose y component is p_1. (One of the ways of performing Operation ℓ_1.)

Table A. 2 (cont.)

Text Statements

```
VISIBLE   100   100   P
INSERT    P     *
```

The following statements are suitable for the state of this program during which the line is rotated:

```
GENERATOR   1   *   P
MEASURE     P   X   Y
ASSIGN      X   -Y/10+X
ASSIGN      Y   X/10+Y
VISIBLE     Y   X   R
SUBSTITUTE  *   1   P   R
CHEW        P
```

These statements measure the components of the current vector, generate a new vector which is a rotated form of this original vector, substitute the new vector for the old one, and destroy the old vector. (Note that the ordering of the terms in the ASSIGN statements is important because of the left precedence rule stated above.) Finally, the statement

```
TYPE    "END"
```

is associated with the final state of this program to complete the program specification.

A. 2. 3 An Example of an Application Program

The example program shown in Figure A. 3, while instructive because of its simplicity, does not illustrate much use of the topological structure. Furthermore, this example does not illustrate the use of very many of the text statements. For these reasons, a second example, which is a modification of the graph theory application described in Chapter 5, is given below.

Note that there is no text statement in Table A. 2 which performs Operation ℓ_{14}. However, according to Table 5. 9, Operation ℓ_{14} is needed in order to respond to input k_5. While other methods may be used to respond to k_5 without using Operation ℓ_{14}, these methods are rather complicated. Consequently, the graph theory program described here is assumed to respond to input k_5 by moving only vertices which are not connected to other vertices by edges. The responses to other inputs are those described in Chapter 5.

The inputs $k_1, k_2, \ldots, k_6$ will be assumed to be generated as follows:

k_1: Point to INK light button with light pen to create a vertex.

k_2: Point to vertex to be destroyed and generate P(↑) input.

k_3: Point to one vertex to be connected to a second vertex by a new edge and generate P(*) input. A mark (the character "/") will appear on this vertex. Now point to the second

vertex. The mark is now removed and the new edge is created.

k_4: Point to edge to be destroyed and generate P(↑) input.

k_5: Point to vertex to be moved and generate P(←) input. The vertex (if not joined to other vertices by edges) will move with the light pen until tracking is lost.

k_6: Point to edge to be bent and generate P(←) input. A point on the edge will move with the light pen until tracking is lost.

The inputs k_5 and k_6 have been interpreted loosely here as inputs which start continuous movement of objects. These inputs as defined in Chapter 5 are actually the loss of tracking after each of the objects has been moved.

The program whose state diagram is shown in Figure A.5 and whose text statements are shown in Table A.3 provides the desired responses to these inputs. In Figure A.5, the states have been numbered to facilitate discussion. These numbers do not actually appear when the diagram is drawn on the display screen. In Table A.3, these numbers are used to associate text statements with the various statements.

The program begins execution in State 1 by creating the light button "INK" and the vertex symbol class in E_0/D_0. It then proceeds

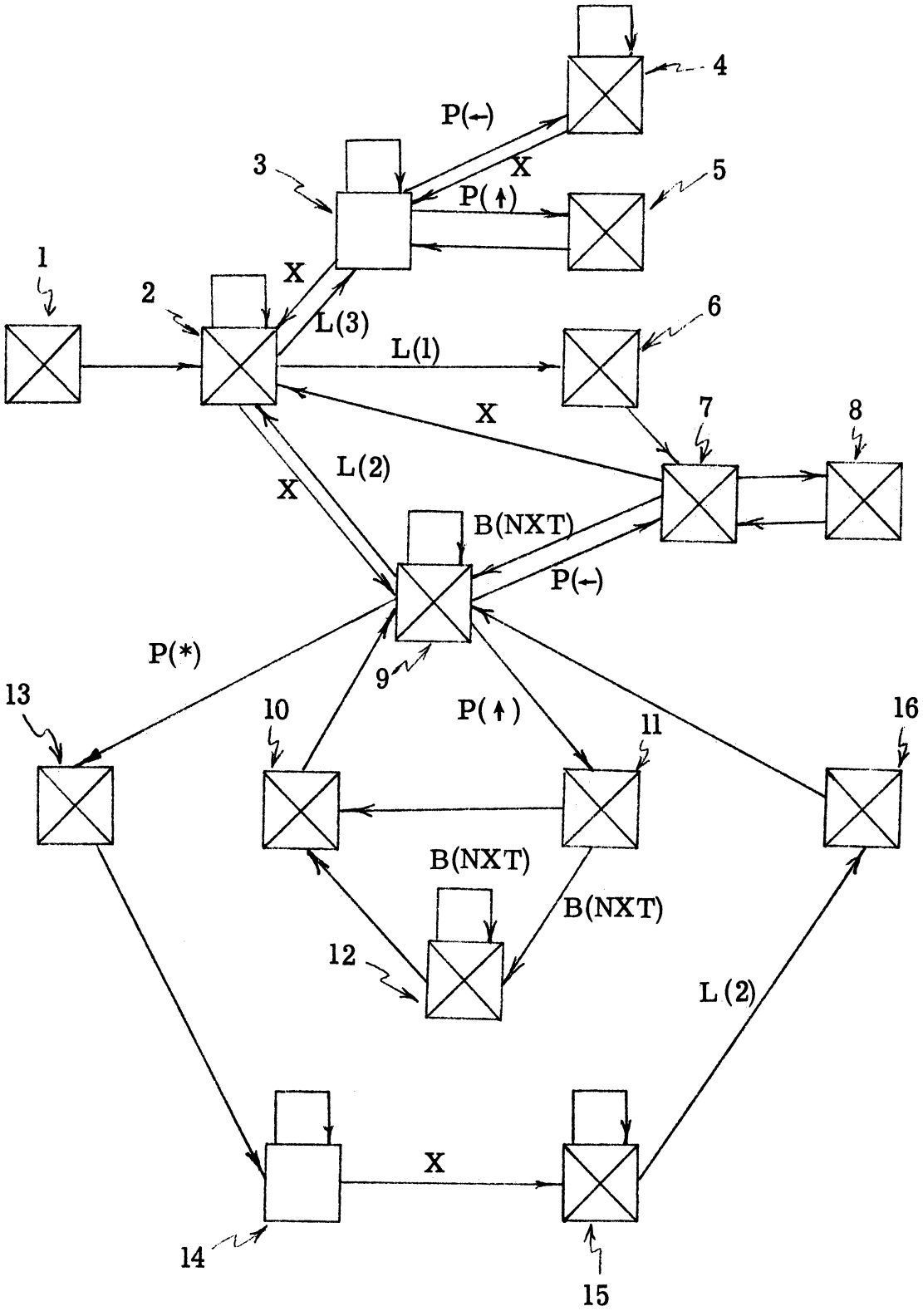


Figure A. 5

State Diagram for Graph Theory Program

State	Statements
1	NONPRIMITIVE N DEFINE T "INK" INSERT T N HOMOGENIZE N T CHEW N INSERT T * EXPOSE T 1 NONPRIMITIVE N BLANK -32 0 T INSERT T N VISIBLE 64 0 T INSERT T N BLANK -32 -32 T INSERT T N VISIBLE 0 64 T INSERT T N BLANK 0 -32 T INSERT T N HOMOGENIZE N V CHEW N EXPOSE V 2
2	UNBLOCK
4	QUERY 3 1 N GENERATOR 2 N] LOCATE 1 T XP Y FIND 0 X Y MEASURE T XT YT ASSIGN XT XT-X+XP ASSIGN YT YT-Y+YP VISIBLE YT XT R EXPOSE R 3 SUBSTITUTE N 1 T R CHEW T

Table A.3

Text Statements for Graph Theory Program

State	Statements
4 (cont.)	GENERATOR 1 N T MEASURE T XT YT ASSIGN XT XT+X-XP ASSIGN YT YT+Y-YP VISIBLE YT XT R EXPOSE R 3 SUBSTITUTE N 1 T R CHEW T
5	QUERY 3 1 N OWNER 1 N T REMOVE T 1 N OWNER 1 N T REMOVE T 1 N CHEW N
6	FIND 1 X Y NONPRIMITIVE N INSERT V N BLANK Y X T INSERT T N INSERT N * 0 0 1
7	GENERATOR 3 N NXT
8	GENERATOR 1 N T FIND 0 X Y BLANK Y X R SUBSTITUTE N 1 T R CHEW T
9	QUERY 2 1 N
10	CHEW N

Table A.3 (cont.)

Text Statements for Graph Theory Program

State	Statements
11	REMOVE * 1 N REMOVE N 1 V ASSIGN I 2 GENERATOR 2 N NXT
12	OWNER 1 NXT T REMOVE T 1 NXT ASSIGN I I+1 GENERATOR I N NXT
13	DEFINE T "/" INSERT T N 1 V 1 GENERATOR 1 N TP MEASURE TP X Y
15	UNBLOCK
16	REMOVE N 1 T CHEW T QUERY 2 1 NP NONPRIMITIVE N INSERT NN N 1 V 1 INSERT NN NP 1 V -1 GENERATOR 1 NP T MEASURE T XP YP ASSIGN X XP-X/2 ASSIGN Y YP-Y/2 VISIBLE Y X T VISIBLE Y X TP INSERT T NN INSERT TP NN EXPOSE T 3 EXPOSE TP 3

Table A.3 (cont.)

Text Statements for Graph Theory Program

to State 2, where it enables the light pen and waits for a light pen input. Since no vertices or edges are being displayed, the light pen input which eventually occurs is an input on the light button "INK", which causes the program to enter State 6. In State 6, the program creates a vertex at the coordinates of the light pen hit and proceeds to State 7. It then alternates between States 7 and 8 to move the vertex until tracking is lost. When tracking is lost, the program returns to State 2 to wait for another input.

After several vertices are created in this way, the user aims the light pen at one of them. The program then enters State 9, where it immediately identifies the vertex and waits for a tracking pattern to be recognized. If the user removes the pen without generating a tracking pattern, the program returns to State 2 without modifying the vertex. If he generates a P(←) input, the program enters State 7 and moves the vertex as it did just after the vertex was created. If he generates a P(↑) input, the program enters State 11. In State 11, the program sets NXT to nonzero if there are edges to be deleted, or zero otherwise. If NXT is nonzero, the program proceeds to State 12, where it deletes all of the edges which were connected to the vertex. It then proceeds to State 10, where it destroys the remainder of the vertex, and then to State 9. The program waits in State 9 for tracking to be lost, since the only other exits from State 9

require tracking pattern inputs, which are now prohibited (because one of them has occurred) until tracking is lost.

If, instead of a P(↑) input, the user generates a P(*) input, the program proceeds to State 13, where it marks that vertex, and then to State 14, where it waits for tracking to be lost. After tracking is lost, the program proceeds to State 15, where it enables the light pen and waits for a light pen hit on another vertex. When this light pen hit occurs, the program proceeds to State 16, where it generates the edge, and then to State 9, where it waits for loss of tracking.

After several edges have been created in this way and the program is again waiting in State 2, the user may aim the light pen at an edge. The program then proceeds to State 3, where it waits for a tracking pattern input. If the user generates a P(↑) input, the program proceeds to State 5, where it deletes the edge, and then returns to State 3 to wait for loss of tracking. If, instead of a P(↑) input, the user produces a P(←) input, the program proceeds to State 4, where it bends the edge. The program remains in this state until tracking is lost.

A.3 Operation of the Program

The program which is described here is self-contained. It should be loaded from paper tape via hardware read-in mode, starting at location 0. The program is also self-starting. When the program

starts, it waits for one of the following commands from the teletype:

LOAD	Load a binary paper tape previously prepared by this program.
PUNCH	Punch a binary tape of the application program which is currently in memory.
RUN	Run the application program which is currently in memory.
UNCOMPILE	Generate a state diagram for the program which is currently in memory.

(Only the first character of any one of these commands is needed. The program will complete the command.)

Initially, the program which resides in memory is a null program. However, if the UNCOMPILE command is issued, light buttons will appear on the screen to enable a state diagram to be drawn. These light buttons are interpreted as follows:

CREATE	A state symbol is created and moved with the light pen until tracking is lost.
CONNECT	The program enters a connect state during which transitions may be drawn with the light pen. This state is terminated via an ESCAPE light button.
DEVICE	A list of device codes is displayed. Upon

selecting one of these device codes with the light pen, the user should type the character for that device on the keyboard. The light pen should then be aimed at a state transition symbol to either add or change a label.

TRANSLATE — The program enters translate state during which the entire diagram may be moved by aiming the light pen at a state symbol. This state is terminated via an ESCAPE light button.

COMPILE — The program asks the user to indicate the initial state with the light pen, and then it compiles the program. The LOAD, PUNCH, RUN, and UNCOMPILE teletype com mands are then enabled.

State symbols are created via the CREATE light button. Once a state symbol is created, it may be deleted via a P(↑) input or moved via a P(←) input. If a P(*) input is given on a state symbol, the state diagram is compiled, and the text statements associated with that state are uncompiled and displayed on the screen. (Initially, no text statements are associated with a state symbol.) The user may then insert or delete text statements via the teletype.

To facilitate the insertion or deletion of text statements, a cursor is provided on the screen. Whenever the cursor is not visible, it is positioned on an imaginary line just before the first line of text. The line feed character advances the cursor on the statements which are being displayed. If the cursor is advanced past the last statement, it is again positioned on the imaginary line just before the first line of text.

In order to insert a statement, the user types the first character of its keyword (which is completed by the program), followed by the parameters for that statement. The statement is then inserted just below the cursor, and the cursor is advanced. In order to delete a statement, the user positions the cursor on that statement (via the line feed character) and types a rubout character. The user returns to the state diagram via the ESCAPE light button.

State transitions are created via the CONNECT light button described above. They are labeled via the DEVICE light button described above. Once a state transition is created, it may be deleted via a P(↑) input, or bent via a P(←) input.

Once the program is compiled (via the COMPILE light button described above), it may be punched (via the PUNCH teletype command), run (via the RUN teletype command), or again uncompiled. Whenever a program is run, it may be terminated at any time via the manual

interrupt button. The LOAD, PUNCH, RUN, and UNCOMPILE commands are then again enabled.

A.4 Suggested Improvements

As has been mentioned in Section A.1 above, a means of representing a sequence $\gamma(\rho(e,e')) = (G(D[\,e'\,])^{-1}, G(D[\,e'\,]), G(D[\,e'\,])^{-1})$ should be included in an implementation of the structure $\{B, E/D, \mu, \gamma\}$. This program, then, should be extended to allow the use of elements of B which are represented by blocks with the format shown in Figure A. 2b.

The display language also could be improved. Since the purpose of this program was to illustrate the structure, rather than the language, not much thought was given to the language. However, one shortcoming of the language which is closely related to the structure is that there is no facility to store pointers to elements of B. In order to designate an element of B, the user must supply such parameters as "the i th element of B whose first component is p_1", or "the i th element of B whose first component is p_1 and whose second component is p_2". Not only is this specification clumsy, but, in more complicated programs, the value of i for such a specification may be difficult to determine.

Another difficulty with the display language used here is that not all of the basic operations can be performed easily. In particular, Operations ℓ_5, ℓ_7, ℓ_9, and ℓ_{14} cannot be performed easily. As was

demonstrated, the inability to perform Operation ℓ_{14} so greatly complicated the process of responding to one of the inputs that this response was considered impractical to program.

Finally, some of the features of the language itself could be improved. With a little more effort, one could eliminate the JUSTIFY statement by defining an absolute value operator to be included in expressions. Furthermore, a more standard expression scanner could be written so that a predefined precedence could be assigned to each operator and could be altered through the use of parentheses. These refinements, however, are not as significant as those improvements which affect the use of the structure, for the expressions which appear in application programs which are prepared with this program are generally rather simple.

Appendix B

PROGRAMS USED FOR ANALYSIS

Four programs were prepared for the IBM 360/67 under MTS [40] to compare implementations of topological structures for various applications of various hardware configurations. One of these programs (OPT) enumerates the implementations for which $g < t_f$ for a given set of parameter values and sorts these implementations according to cost. The output of this program is not printed, but it is saved on files so that it may be further processed. The other three programs (PRINT, CON, and PLOT) are post-processors which operate on the data in these files to print the tables, to apply the constraints $\underline{s} \leq s_{max}$ and $t_f \leq t_{max}$, and to plot the cost of selected implementations versus parameters which describe the application (e.g., n_c and n_ℓ for the text editor described in Chapter 5). The behavior of each of these programs is described below.

B.1 Enumeration

The program OPT, which enumerates implementations for which $g < t_f$ and sorts these implementations according to cost, was designed to be run from a teletype, rather than in batch mode. This program contains a command language interpreter which accepts the following commands:

ALTER: Alter the values of parameters whose symbols are inputted on the teletype to values inputted on the teletype.

CALCULATE: Call the subroutine PAR, which must be supplied by the user, to evaluate some or all of the parameters which describe the application and hardware configuration.

LIST: Type a list of the current values of the parameters which describe the application and hardware configuration.

READ: Read values of parameters which describe the application and hardware configuration from a file which has been produced by either a combination of ALTER and SAVE commands or the SOLVE command.

SAVE: Save the current values of the parameters which describe the application and hardware configuration on a specified file.

SOLVE: Enumerate the implementations for the current values of the parameters which describe the application and hardware configuration for which $g < t_f$, sort these implementations according to cost, and output tables of results on a specified file.

Since the first two letters of each of these commands distinguishes it from all other commands, only the first two letters of each command must actually be typed.

The first five of these commands serve only to aid the input of parameter values and do not contribute to the enumeration of implementations. However, the SOLVE command applies the equations stated in Chapters 3 and 4 to compute values of s, g, t_f and $\bar{r}$. A simplified flow chart of the response to the solve command is shown in Figure B.1. The applicable equations from Chapters 3 and 4 are also indicated in this figure.

The details needed to run this program are best illustrated by example. In order to illustrate these details, the operation of this program in order to obtain the results plotted in Figure 5.3 is shown below. In the sequences of teletype information shown, the characters which are typed by the user are underlined, whereas those typed by the program are not underlined.

In order to run the program for the text editor application of the DEC 339, the user might type the following command:

$RUN OPT+ EDIT

The file EDIT in this example contains the object program for the subroutine PAR which evaluates those parameters whose values depend on the application which is being considered. A suitable

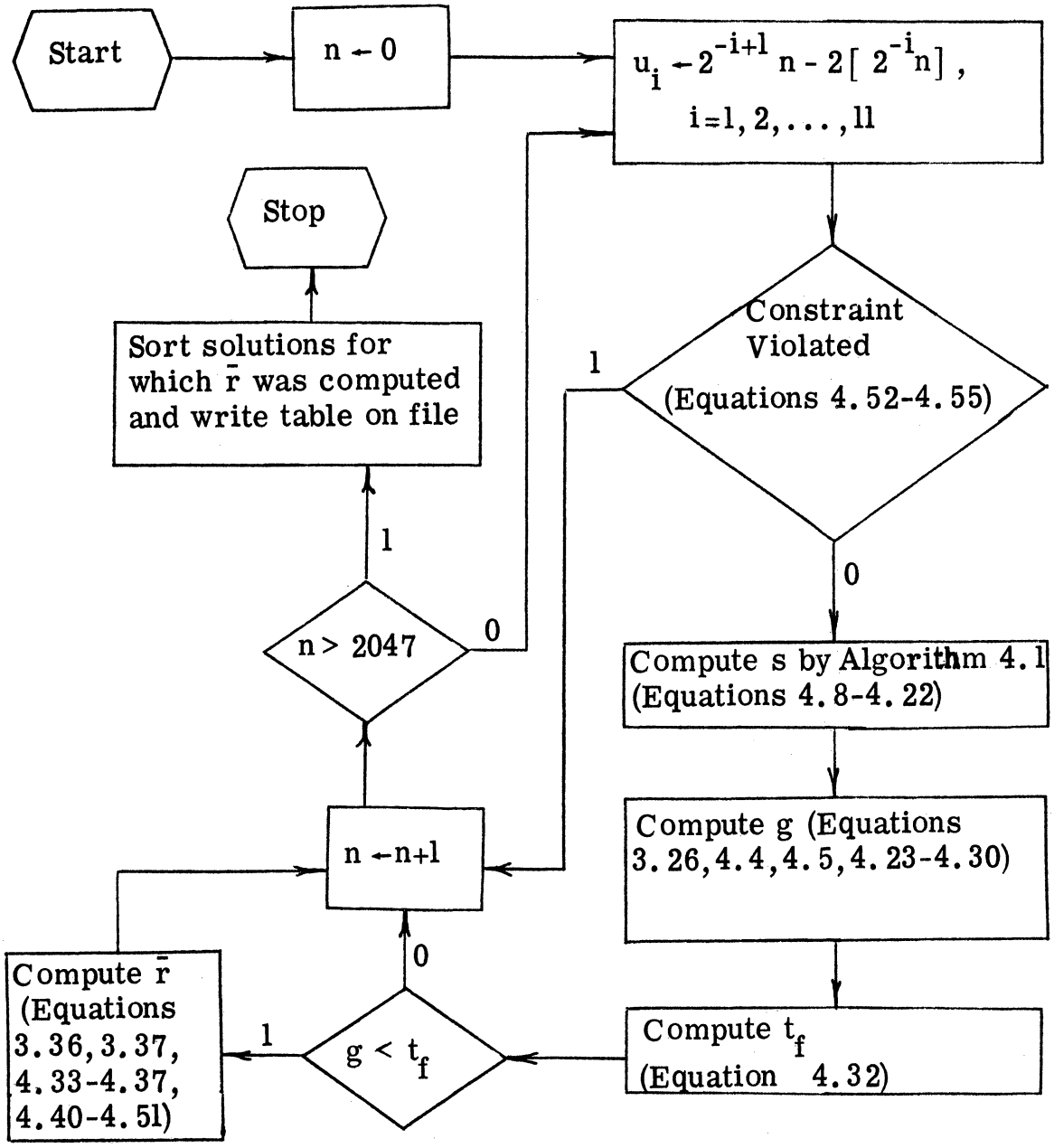


Figure B.1

Flow Chart for the SOLVE Command

FORTRAN program for the subroutine PAR for this example is shown in Figure B. 2. The COMMON statement in this program contains a list of variables which correspond to the parameters which describe the application and hardware configuration. These variable names, however, are not necessarily the same as those which are inputted with the ALTER command, and are not the names used in the text. The correspondences between these various types of notation are shown in Table B.1. The binary - valued variables SW1 and SW2 in the COMMON statement describe whether or not the parameters S_γ and S_γ', respectively, are averages of integer numbers of locations. If one of these variables is a 0, the corresponding parameter is the average of integer numbers of locations; otherwise, it is not.

When execution of the program begins, the value of every parameter is set to zero. The program then waits for a command. Suppose now that the user wishes to input all of the parameters which are peculiar to the DEC 339 hardware configuration, but which are independent of the application. He then types the following sequence:

```
AL
=.005,TP
=.002,TJ
=.004,TS
=.007,TJ'
=.007,TS'
=.008,TB
```

```
      SUBROUTINE PAR
      IMPLICIT REAL(N)
      INTEGER SW1, SW2
      REAL FF(15),FG(15)
      COMMON BF,BG,EDF,EDG,EDOF,EDOG,
     1       SUMF,SUMG,SUMOF,SUMOG,TDF,TDG,
     1       TP,TJ,TS,TJ1,TS1,TB,TC,SO,SP,SS,SJ,SA,
     1       ST,ST1,SR,SH,SBF,SBF1,
     1       SBG,SBG1,SF,SF1,SG,SG1,SGAM,SGAM1,GS,GS1,GS2,
     1       GS3,GMF,GMF1,GMG,GMG1,GMG2,GHF,GHF1,GHG,GHG1,
     1       GHG2,GBF,GBG,GTF,GTF1,GTG,GTG1,GCF,GCG,RGBAR,
     1       RC,RDBAR,RP,RJ,RSF,RSF1,RSF2,RSG,RSG1,RSG2,
     1       RG,RG1,RF,RF1,RGAM,RGAM1,SIGMA,TTF,TTG,FF,FG,
     1       SW1,SW2
      WRITE(6,1)
1     FORMAT('ENTER NL, NC:')
      READ(5,2) NL,NC
2     FORMAT(2F10.0)
      BF=NL*NC+NL
      BG=NL*NC+3.*NL+1.
      EDF=NL+65.
      EDG=2.*NL+67.
      EDOF=64.
      EDOG=66.
      SUMF=BF
      SUMG=BG
      SUMOF=NL*NC
      SUMOG=NL*NC+NL+1.
      SF1=6.-6.*NL/BF
      SBF1=4.
      IS=0
      INL=NL
      DO 100 I=1,INL
100   IS=IS+IABS(I-20)
      TTF=.0024*NL*NC*(NC-1.)+.1536*NL+.0072*IS
      TDF=.02*NL*NC
      GBF=(.012*EDF-.004*EDOF)/SUMF
      GMF1=.003-.003*NL/SUMF
      GHF1=GMF1
      RF1=.094-.082*NL/BF
      GTF1=.004
```

Figure B.2

PAR Subroutine for Text Editor

```
SGAM1=4.-2.*NL/BG
SW2=0
SBG1=2.015
TTG=.0024*NL*NC
TDG=.02*NL*NC+.0036*NL+.1536
GBG=(.012*EDG-.004*EDOG)/SUMG
GMG=.006
GMG2=.006
GHG=(.05+.067*NC)*NL/SUMG
GHG2=(.05+.054*NC+.00020312 5*NL*NC)*NL/SUMG
GHG1=(.009+.012*NC)*NL/SUMG
RGAM1=.012+.018*NL/BG
GTG1=(.003+.002*(NL*NC+NL))/SUMG
FF(1)=.25
FF(2)=.25
FF(3)=.5+.25*NL+.25*NC
FF(4)=.5+.25*NL+.5*NC
FF(6)=.25
FF(10)=.25*NL+.5*NC
FF(12)=.25*NL+.25*NC
FF(13)=.25*NL+.25*NC
FG(1)=.5
FG(2)=.5
FG(3)=1.
FG(4)=1.+.25*NC
FG(6)=.25
FG(10)=.25*NC
RETURN
END
```

Figure B. 2 (cont.)

PAR Subroutine for Text Editor

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
q_{bf}	BF	BF
q_{bg}	BG	BG
q_{df}	EDF	EDF
q_{dg}	EDG	EDG
q_{d0f}	ED0F	ED0F
q_{d0g}	ED0G	ED0G
q_{sf}	SUMF	SUMF
q_{sg}	SUMG	SUMG
q_{s0f}	SUM0F	SUM0F
q_{s0g}	SUM0G	SUM0G
s_0	S0	S0
S_t	ST	ST
S_t'	ST'	ST1
S_p	SP	SP

Table B.1

Notation for Parameters

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
S_s	SS	SS
S_j	SJ	SJ
S_a	SA	SA
S_r	SR	SR
S_h	SH	SH
S_f	SF	SF
S_f'	SF'	SF1
$S_{\beta f}$	SBF	SBF
$S_{\beta f}'$	SBF'	SBF1
S_g	SG	SG
S_g'	SG'	SG1
S_γ	SGAM	SGAM
S_γ'	SGAM'	SGAM1
$S_{\beta g}$	SBG	SBG
$S_{\beta g}'$	SBG'	SBG1

Table B.1 (cont.)

Notation for Parameters

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
T_p	TP	TP
T_j	TJ	TJ
T_s	TS	TS
T_j'	TJ'	TJ1
T_s'	TS'	TS1
T_b	TB	TB
T_c	TC	TC
T_{tf}	TTF	TTF
T_{df}	TDF	TDF
T_{tg}	TTG	TTG
T_{dg}	TDG	TDG
G_s	GS	GS
G_s'	GS'	GS1
G_s''	GS''	GS2
G_s'''	GS'''	GS3

Table B.1 (cont.)

Notation for Parameters

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
G_{mf}	GMF	GMF
G_{mf}'	GMF'	GMF1
G_{hf}	GHF	GHF
G_{hf}'	GHF'	GHF1
G_{mg}	GMG	GMG
G_{mg}'	GMG'	GMG1
G_{mg}''	GMG''	GMG2
G_{hg}	GHG	GHG
G_{hg}'	GHG'	GHG1
G_{hg}''	GHG''	GHG2
G_{bf}	GBF	GBF
G_{bg}	GBG	GBG
G_{cf}	GCF	GCF
G_{tf}	GTF	GTF
G_{tf}'	GTF'	GTF1

Table B.1 (cont.)

Notation for Parameters

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
G_{cg}	GCG	GCG
G_{tg}	GTG	GTG
G_{tg}'	GTG'	GTG1
$\bar{r}_g$	RG-	RGBAR
r_c	RC	RC
$\bar{r}_d$	RD-	RDBAR
R_p	RP	RP
R_j	RJ	RJ
R_f	RF	RF
R_f'	RF'	RF1
R_g	RG	RG
R_g'	RG'	RG1
R_γ	RGAM	RGAM
R_γ'	RGAM'	RGAM1
R_{sf}	RSF	RSF

Table B.1 (cont.)

Notation for Parameters

Symbol Used in Text	ALTER Command Symbol	COMMON Statement Symbol
R_{sf}'	RSF'	RSF1
R_{sf}''	RSF''	RSF2
R_{sg}	RSG	RSG
R_{sg}'	RSG'	RSG1
R_{sg}''	RSG''	RSG2
σ	SIGMA	SIGMA
$f_f(\ell_i)$	FF(i)	FF(i)
$f_g(\ell_i)$	FG(i)	FG(i)

Table B.1 (cont.)

Notation for Parameters

=.005,TC

=2,SO

=.833333,SP

=1.666667,SS

=1.666667,SJ

=1.666667,SA

=.111111,ST

=.111111,ST'

=.666667,SR

=.666667,SH

=2,SF

=2,SG

=4,SG'

=.111111,SGAM

=DOES EACH SEQUENCE REQUIRE THE SAME AMOUNT OF STORAGE?

=YES

=.01,GS

=.003,GS'

=.02,GS''

=99999999999999,GS'''

=.016,GMF

=.022,GHF

=.15,RG-

=.006,RC

=.15,RD-

=.005,RP

=.046,RJ

=.012,RG

=.082,RG'

=.012,RF

=.007,RGAM

=5,SIGMA

In order to terminate the ALTER command, the user types an end-of-file. Note that when a value of S_γ is inputted, the program requests information necessary to set SW1 to the proper value.

Now suppose the user wishes to type a list of the current values of the parameters and to save them on the file DEC 339 for future use. He first issues the command

LI

to produce the listing of these parameters, which is shown in Figure B.3. After verifying that these values are correct, the user saves the parameter values on the file DEC 339 as follows:

SA

ENTER FILE NAME:

DEC339

These parameter values may be restored at a later time as follows:

RE

ENTER FILE NAME:

DEC339

BF	=	0.0	BG	=	0.0	EDF	=	0.0
EDG	=	0.0	EDOF	=	0.0	EDOG	=	0.0
SUMF	=	0.0	SUMG	=	0.0	SUMOF	=	0.0
SUMOG	=	0.0	TDF	=	0.0	TDG	=	0.0
TP	=	0.005	TJ	=	0.002	TS	=	0.004
TJ'	=	0.007	TS'	=	0.007	TB	=	0.008
TC	=	0.005	SO	=	2.000	SP	=	0.833
SS	=	1.667	SJ	=	1.667	SA	=	1.667
ST	=	0.111	ST'	=	0.111	SR	=	0.667
SH	=	0.667	SBF	=	0.0	SBF'	=	0.0
SBG	=	0.0	SBG'	=	0.0	SF	=	2.000
SF'	=	0.0	SG	=	2.000	SG'	=	4.000
SGAM	=	0.111	SGAM'	=	0.0	GS	=	0.010
GS'	=	0.003	GS''	=	0.020	GS'''	=	**********
GMF	=	0.016	GMF'	=	0.0	GMG	=	0.0
GMG'	=	0.0	GMG''	=	0.0	GHF	=	0.022
GHF'	=	0.0	GHG	=	0.0	GHG'	=	0.0
GHG''	=	0.0	GBF	=	0.0	GBG	=	0.0
GTF	=	0.0	GTF'	=	0.0	GTG	=	0.0
GTG'	=	0.0	GCF	=	0.0	GCG	=	0.0
RG-	=	0.150	RC	=	0.006	RD-	=	0.150
RP	=	0.005	RJ	=	0.046	RSF	=	0.0
RSF'	=	0.0	RSF''	=	0.0	RSG	=	0.0
RSG'	=	0.0	RSG''	=	0.0	RG	=	0.012
RG'	=	0.082	RF	=	0.012	RF'	=	0.0
RGAM	=	0.007	RGAM'	=	0.0	SIGMA	=	5.000
TTF	=	0.0	TTG	=	0.0	FF(1)	=	0.0
FF(2)	=	0.0	FF(3)	=	0.0	FF(4)	=	0.0
FF(5)	=	0.0	FF(6)	=	0.0	FF(7)	=	0.0
FF(8)	=	0.0	FF(9)	=	0.0	FF(10)	=	0.0
FF(11)	=	0.0	FF(12)	=	0.0	FF(13)	=	0.0
FF(14)	=	0.0	FF(15)	=	0.0	FG(1)	=	0.0
FG(2)	=	0.0	FG(3)	=	0.0	FG(4)	=	0.0
FG(5)	=	0.0	FG(6)	=	0.0	FG(7)	=	0.0
FG(8)	=	0.0	FG(9)	=	0.0	FG(10)	=	0.0
FG(11)	=	0.0	FG(12)	=	0.0	FG(13)	=	0.0
FG(14)	=	0.0	FG(15)	=	0.0			
SW = 1		SW' = *						

Figure B.3

Listing of Parameters for DEC 339

The user now wishes to produce the data necessary to obtain Figure 5.3. In order to request enumeration of the implementations for $n_\ell = 20$ and $n_c = 5$, he engages in the following teletype conversation:

CA

ENTER NL,NC:

20,5,

SO

ENTER FILE NAME:

RESULTS

TITLE:

EDITOR NL =20 NC =5

ENTER 0 MASK:

0001

ENTER 1 MASK:

01

The request to enter n_ℓ and n_c is typed by the subroutine PAR, which is called in response to the CALCULATE command. In response to the SOLVE command, the program requests the name of the file on which results are to be outputted and a title to be outputted with these results so that they can be identified. The program then requests two 11-bit masks. The 0 mask contains 1's in those positions which correspond to design decisions which are to be constrained to assume

the value 0, whereas the 1 mask contains 1's in those positions which correspond to design decisions which are to be constrained to assume the value 1. (Bits in these masks which are not specified are assumed to be zeros.) In this example, u_4 is constrained to be zero and u_2 is constrained to be 1.

In order to compute the other information needed for Figure 5.3, the user types CALCULATE and SOLVE commands to tabulate implementations for other values of n_c. (Note that these subsequent results may be appended to the file RESULTS by specifying RESULTS(LAST+1) if RESULTS is a line file, or simply RESULTS if RESULTS is a sequential file.) The user terminates the program by typing an end-of-file in lieu of a command. To permit flexibility of operation, the program was written so that it may be restarted (via the $RESTART command) after it has been terminated in this way.

B.2 Printing

Because the program OPT produces a large amount of output, this output is written on files in binary format. The program PRINT was written to read these files and print the information which they contain in readable format. PRINT was designed to be run in batch mode. The program reads from data set number 0 and writes on data set number 6. Consequently, the MTS command

$RUN PRINT 0 = RESULTS

will print all of the tables of results prepared by the example run of OPT described above.

B.3 Application of Constraints

The user often desires to vary s_{max} and t_{max} for a particular application of a particular hardware configuration. Consequently, the constraints $s \leq s_{max}$ and $t_f \leq t_{max}$ are not applied by the program OPT. Instead, the tables which OPT produces contain values of s and t_f, and these constraints are applied by the post-processing program CON.

CON is designed to be run from a teletype. In order to start this program, the user issues the MTS command

$RUN CON

The program begins execution by waiting for a command from the user. The commands which are interpreted are the following:

READ	A file of results prepared by OPT is read for consideration.
TEST	Values of s_{max} and t_{max} are accepted from the user and the optimum implementation in each table in the file of results being considered which satisfies the constraints $s \leq s_{max}$ and $t_f \leq t_{max}$ in every table in the file is identified to the user.

As in the program OPT, only the first two characters of either of

these commands must be typed.

In order to illustrate this program, its application to a file GRAPH which contains data for the graph theory application described in Chapter 5 for $\ell = 100$, $n_e = 100$, and $n_v = 20$, 25, 35, 50, 65, 80, 100, 120, and 140 is considered. The following teletype conversation identifies the optimum implementations for each value of n_v with $s_{max} = 24000$ locations, $t_{max} = 500$ msec, and $u_1 = 1$.

RE

ENTER FILE NAME:

GRAPH

TE

ENTER, SMAX, TMAX:

24000, 500,

ENTER 0 MASK:

0

ENTER 1 MASK:

1

OPTIMUM SOLUTION IN EACH TABLE:

1767 1767 707 707 707 707 707 707 707

(The 0 and 1 masks have the same meaning here as in the program OPT.) Each solution is identified here as the decimal equivalent of the binary number $u_{11}\ u_{10} \ldots u_1$. These decimal numbers will be

called the names of the various implementations. Thus, 1767 is the name of the implementation 11100111011 and 707 is the name of the implementation 11000011010. (Note that the binary representation of each implementation is $u_1\ u_2 \ldots u_{11}$, rather than $u_{11}\ u_{10} \ldots u_1$.)

B.4 Plotting

Figures 5.3, 5.4, 5.6, 5.7, and 5.8 were prepared by the program PLOT. This program was designed to be run from a teletype. It allows the user to plot the cost of up to 16 implementations per graph versus an application parameter. The plot description is written on data set number 9 for later post-processing under MTS [20]. For example, this program may be applied to plot the cost of the implementations 1767 and 707 from the file GRAPH used in the example given in Section B.3. The teletype conversation to generate this plot would be the following:

$RUN PLOT 9 =PLOTQUE

ENTER FILE NAME:

GRAPH

ENTER ABSCISSA VALUES:

20, 25, 35, 50, 65, 80, 100, 120, 140,

ENTER ABSCISSA LABEL:

NUMBER OF VERTICES

ENTER NUMBER OF CHARACTERS IN THIS LABEL:

18

ENTER CURVE NAMES:

1767, 707,

ENTER CURVE NAMES:

As indicated, the program will continue to request curve (i.e., implementation) names after a list of names has been given. Each list of curve names is plotted on a separate graph. If the user desires to read a new file when the program requests more curve names, he should type a null line. If, however, he wishes to terminate the program, he should type an end-of-file.

BIBLIOGRAPHY

[1] 339 Programmed Buffered Display, DEC-09-I6FA-D, Digital Equipment Corporation, Maynard, Massachusetts, May 1968.

[2] Allen, T. R., and J. E. Foote, "Input/Output Software Capability for a Man-Machine Communication and Image Processing System", Proceedings of the Fall Joint Computer Conference, 1964, pp. 387-396.

[3] Baecker, R., "Picture-Driven Animation", Proceedings of the Spring Joint Computer Conference, 1969, pp. 273-288.

[4] Ball, N. A., et al., "A Shared Memory Computer Display System", IEEE Transactions on Electronic Computers, Vol. EC-15, No. 5, October, 1966, pp. 750-756.

[5] Baskin, H. B., and S. P. Morse, "A Multilevel Modeling Structure for Interactive Graphic Design", IBM Systems Journal, Vol. 7, Nos. 3 and 4, 1968, pp. 218-228.

[6] Boehm, B. W., et al., "POGO: Programmer-Oriented Graphics Operation", Proceedings of the Spring Joint Computer Conference, 1969, pp. 321-330.

[7] Bond, A. H., et al., "An Interactive Graphical Display Monitor in a Batch-Processing Environment with Remote Entry", Communications of the ACM, Vol. 12, No. 11, November 1969, pp. 595-603.

[8] Brenner, A. E., and P. de Bruyne, "A Sonic Pen: A Digital Stylus System", IEEE Transactions on Computers, Vol. C-19, No. 6, June 1970, pp. 546-548.

[9] Calvert, T. W., "Projections of Multidimensional Data for Use in Man-Computer Graphics", Proceedings of the Fall Joint Computer Conference, 1968, pp. 227-231.

[10] Cameron, S. H., et al., "DIALOG: A Conversational Programming System with a Graphical Orientation", Journal of the ACM, Vol. 10, No. 6, June 1967, pp. 349-357.

[11] Chen, F. C., and R. L. Dougherty, "A System for Implementing Interactive Applications", IBM Systems Journal, Vol. 7, Nos. 3 and 4, 1968, pp. 257-270.

[12] Christensen, C., and E. N. Pinson, "Multi-Function Graphics for a Large Computer System", Proceedings of the Fall Joint Computer Conference, 1967, pp. 697-711.

[13] Cohen, D., and T. M. P. Lee, "Fast Drawing of Curves for Computer Display", Proceedings of the Spring Joint Computer Conference, 1969, pp. 297-307.

[14] Dertouzos, M. L., "PHASEPLOT: An On-Line Graphical Display Technique", IEEE Transactions on Electronic Computers, Vol. EC-16, No. 2, April 1967, pp. 203-209.

[15] Devere, G. S., et al., "The DAC-I System", Datamation, June 1966.

[16] Eastman, C. M., "Representations for Space Planning", Communications of the ACM, Vol. 13, No. 4, April 1970, pp. 242-250.

[17] Enguolo, K. J., and J. L. Hughes, "A General-Purpose Display Processing and Tutorial System", Communications of the ACM, Vol. 11, No. 10, October 1968, pp. 697-702.

[18] Evans & Sutherland Line Drawing System Model 1 System Reference Manual, Evans & Sutherland Computer Corporation, Salt Lake City, Utah, January 1970.

[19] Farmer, N. A., "A Digital Comparator for Use with Computer Displays", IEEE Transactions on Computers, Vol. C-18, No. 3, March 1969, pp. 269-270.

[20] Fronczak, E. J., The University of Michigan Plot Description System and Related Calcomp 780/763 Post-Processing, Technical Report 29, Concomp Project, University of Michigan, Ann Arbor, Michigan, August 1970.

[21] Gott, A. H., et al., "Teaching Heart Function--One Application of Medical Computer Animation", Proceedings of the Spring Joint Computer Conference, 1969, pp. 637-647.

[22] Hamilton, J. A., A Survey of Data Structures for Interactive Graphics, Memorandum RM-6145-ARPA, The Rand Corporation, Santa Monica, California, April 1970.

[23] Hirsch, P. M., et al., "Kinoforms and Digital Holograms", Pertinent Concepts in Computer Graphics, Ed. by M. Faiman and J. Nievergelt, University of Illinois Press, Chicago, Illinois, 1969.

[24] Hodes, L., "A Programming System for the On-Line Analysis of Biomedical Images", Communications of the ACM, Vol. 13, No. 5, May 1970, pp. 279-283.

[25] Irani, K. B., V. L. Wallace, and J. H. Jackson, "Conversational Design of Stochastic Service Systems from a Graphical Terminal", Computer Graphics 70 International Symposium, Brunel University, Uxbridge, Middlesex, England, April 15, 1970.

[26] Jackson, J. H., An Executive System for a DEC 339 Computer Display Terminal, Concomp Project Technical Report 15; also Systems Engineering Laboratory Technical Report 32, University of Michigan, Ann Arbor, Michigan, December 1968.

[27] Jackson, J. H., SELMA: A Conversational System for the Graphical Specification of Markovian Queueing Networks, Technical Report 23; Concomp Project, University of Michigan, Ann Arbor, Michigan, October 1969.

[28] Johnson, T. E., "SKETCHPAD III - A Computer Program for Drawing in Three Dimensions", Proceedings of the Spring Joint Computer Conference, 1963, pp. 347-353.

[29] Joyce, J. D., and M. J. Cianciolo, "Reactive Displays: Improving Man-Machine Graphical Communication", Proceedings of the Fall Joint Computer Conference, 1967, pp. 713-721.

[30] Knuth, D. E., Fundamental Algorithms, Addison-Wesley Publishing Col, Reading, Massachusetts, 1968.

[31] Konkle, K. H., "An Analog Comparator as a Pseudo-Light Pen for Computer Displays", IEEE Transactions on Computers, Vol. C-17, No. 1, January 1968, pp. 54-55.

[32] Kubert, B., et al., "The Perspective Representation of Functions of Two Variables", Journal of the ACM, Vol. 15, No. 2, April 1968, pp. 193-204.

[33] Kulsrud, H. E., "A General Purpose Graphic Language", Communications of the ACM, Vol. 11, No. 4, April 1968, pp. 247-254.

[34] Lang, C. A., and J. C. Gray, "ASP--A Ring Implemented Associative Structure Package", Communications of the ACM, Vol. 11, No. 8, August 1968, pp. 550-555.

[35] Lechner, B. J., "Liquid Crystal Displays", Pertinent Concepts in Computer Graphics, Ed. by M. Faiman and J. Nievergelt, University of Illinois Press, Chicago, Illinois, 1969.

[36] Lee, T. M. P., "A Class of Surfaces for Computer Display", Proceedings of the Spring Joint Computer Conference, 1969, pp. 309-319.

[37] Lesem, L. B., "Computer Synthesis of Holograms for 3-D Display", Communications of the ACM, Vol. 11, No. 10, October 1968, pp. 661-674.

[38] Licklider, J. C. R., "A Picture Is Worth a Thousand Words--and Costs", Proceedings of the Spring Joint Computer Conference, 1969, pp. 617-621.

[39] Metzger, R. A., "Computer Generated Graphic Segments in a Raster Display", Proceedings of the Spring Joint Computer Conference, 1969, pp. 161-172.

[40] Michigan Terminal System, (2nd Ed.), University of Michigan Computing Center, Ann Arbor, Michigan, 1967.

[41] Miller, J. C., and C. M. Wine, "A Light Pen for Remote Time-Shared Graphic Consoles", IEEE Transactions on Computers, Vol. C-17, No. 8, August 1968, pp. 799-802.

[42] Miller, J. C., and C. M. Wine, "A Simple Display for Characters and Graphics", IEEE Transactions on Computers, Vol. C-17, No. 5, May 1968, pp. 470-475.

[43] Morrison, R. A., "Graphic Language Translation with a Language Independent Processor", Proceedings of the Fall Joint Computer Conference, 1967, pp. 723-731.

[44] Myer, T. H., and I. E. Sutherland, "On the Design of Display Processors", Communications of the ACM, Vol. 11, No. 6, June 1968, pp. 410-414.

[45] Newman, W. M., "A High-Level Programming System for a Remote Time-Shared Graphics Terminal", Pertinent Concepts in Computer Graphics, Ed. by M. Faiman and J. Nievergelt, University of Illinois Press, Chicago, Illinois, 1969.

[46] Newman, W. M., "A System for Interactive Graphical Programming", Proceedings of the Spring Joint Computer Conference, 1968, pp. 47-54.

[47] Ninke, W. H. "Graphic-1--A Remote Graphical Display Console System", Proceedings of the Fall Joint Computer Conference, 1965, pp. 839-846.

[48] Noll, A. M., "A Computer Technique for Displaying n-Dimensional Hyperobjects", Communications of the ACM, Vol. 10, No. 8, August 1967, pp. 469-473.

[49] Ogden, S., and N. Wadsworth, "On-Line Graphics at The University of Utah", Datamation, November 1969, pp. 159-165.

[50] Ophir, D., et al., "GRAD: The Brockhaven Raster Display", Communications of the ACM, Vol. 11, No. 6, June 1968, pp. 415-416.

[51] Ophir, D., et al., "Three-Dimensional Computer Display", Communications of the ACM, Vol. 12, No. 6, June 1969, pp. 309-310.

[52] PDP-9 User Handbook, F-95, Digital Equipment Corporation, Maynard, Massachusetts, January 1968.

[53] Raffel, J. I., Semiannual Technical Summary Report to the Advanced Research Projects Agency on Graphics, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, December 1965.

[54] Roberts, L. G., "A Graphical Service System with Variable Syntax", Communications of the ACM, Vol. 9, No. 3, March 1966, pp. 173-175.

[55] Roberts, L. G., "Graphical Communication and Control Languages", University of Michigan Engineering Summer Conference Notes on Computer Graphics, 1965.

[56] Roberts, L. G., "Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs", University of Michigan Engineering Summer Conference Notes on Computer Graphics, 1965.

[57] Roberts, L. G., Machine Perception of Three-Dimensional Solids, Lincoln Laboratory Technical Report No. 315, Massachusetts Institute of Technology, Lexington, Massachusetts, May 1965.

[58] Rose, G. A., "'Intergraphic', A Microprogrammed, Graphical-Interface Computer", IEEE Transactions on Electronic Computers, Vol. EC-16, No. 6, December 1967, pp. 773-784.

[59] Rose, G. A., "'Light-Pen' Facilities for Direct View Storage Tubes--An Economical Solution for Multiple Man-Machine Communication", IEEE Transactions on Electronic Computers, Vol. EC-14, No. 4, August 1965, pp. 637-639.

[60] Rougelot, R. S., "The General Electric Computer Color TV Display", Pertinent Concepts in Computer Graphics, Ed. by M. Faiman and J. Nievergelt, University of Illinois Press, Chicago, Illinois, 1969.

[61] Shaw, A. C., "Parsing of Graph-Representable Pictures", Journal of the ACM, Vol. 17, No. 3, July 1970, pp. 453-481.

[62] Shaw, A. C., The Formal Description and Parsing of Pictures, Stanford Linear Accelerator Center Report No. 84, Stanford University, Stanford, California, March 1968.

[63] Siders, R. A., et al., Computer Graphics, American Management Association, New York, New York, 1966.

[64] Sproull, R. F., and I. E. Sutherland, "A Clipping Divider", Proceedings of the Fall Joint Computer Conference, 1968, pp. 765-775.

[65] Sutherland, I. E., "A Head-Mounted Three Dimensional Display", Proceedings of the Fall Joint Computer Conference, 1968, pp. 757-764.

[66] Sutherland, I. E., "Computer Displays", Scientific American, June 1970, pp. 56-81.

[67] Sutherland, I. E., SKETCHPAD: A Man-Machine Graphical Communication System, Lincoln Laboratory Technical Report No. 296, Massachusetts Institute of Technology, Lexington, Massachusetts, January 1963.

[68] Sutherland, I. E., "SKETCHPAD - A Man-Machine Graphical Communication System", Proceedings of the Spring Joint Computer Conference, 1963, pp. 329-346.

[69] Sutherland, W. R., et al., "Graphics in Time-Sharing: A Summary of TX-2 Experience", Proceedings of the Spring Joint Computer Conference, 1969, pp. 629-636.

[70] Sutherland, W. R., The On-Line Graphical Specification of Computer Procedures, Ph.D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Lexington, Massachusetts, January 1966.

[71] System Reference Manual--Adage Graphics Terminal, Adage, Inc., Boston, Massachusetts, March 1968.

[72] Theiss, C. M., "Computer Graphics Displays of Simulated Automobile Dynamics", Proceedings of the Spring Joint Computer Conference, 1969, pp. 289-296.

[73] Uber, G. T., et al., "The Organization and Formatting of Hierarchial Displays for the On-Line Input of Data", Proceedings of the Fall Joint Computer Conference, 1968, pp. 219-226.

[74] Van Dam, A., and D. Evans, "A Compact Data Structure for Storing, Retrieving, and Manipulating Line Drawings", Proceedings of the Spring Joint Computer Conference, 1967, pp. 601-608.

[75] Williams, R., "On the Application of Graph Theory to Computer Data Structures", Computer Graphics 70 International Symposium, Brunel University, Uxbridge, Middlesex, England, April 15, 1970.

[76] Wolfberg, M. S., An Interactive Graph Theory System, Technical Report 69-25, The Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Penn., June 1969.

[77] Wylie, C., et al., "Half-Tone Perspective Drawings by Computer", Proceedings of the Fall Joint Computer Conference, 1967, pp. 49-58.