

*Setting Tables and  
Illustrations with Style*

*Richard J. Beach*

*CS 85-45*

*1985*

# Setting Tables and Illustrations with Style

Richard J. Beach

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirements for the degree of  
Doctor of Philosophy  
in  
Computer Science

Waterloo, Ontario, 1985

© Richard J. Beach 1985

# Setting Tables and Illustrations with Style

---

## Abstract

This thesis addresses the problem of formatting complex documents with electronic tools. In particular, the two problems of incorporating illustrations and laying out tables are treated in depth. The notion of style, a way of maintaining consistency in a document, runs throughout the thesis. It helps manage the complexities of formatting both illustrations and tables. The thesis reviews the history of document composition systems, including early computer typesetting systems, document compilers, and integrated document composition systems. The concept of graphical style is introduced to extend the more traditional notion of document style to illustrations. The observation that graphical style does not adequately deal with the layout problem for illustrations leads to the investigation of a more concentrated layout problem for the special case of table formatting. A grid system is used to describe the table layout arrangement and a constraint solver provides the general layout engine for formatting tables as well as the basis for future interactive table design tools. Further research based on the style and table formatting models can be extended to mathematical typesetting and full page layout. A glossary of typesetting terms and an index to the thesis are provided to help the reader deal with the typographic terminology used in the thesis.

# Contents

---

Abstract	iv
Table of Figures	ix
Acknowledgements	xi
<b>1 Introduction</b>	<b>1-1</b>
1.1 Overview	1-1
1.2 What is a document?	1-2
1.3 Personal Reflections on Document Production	1-5
1.4 The Concept of Document Style	1-9
1.5 Roadmap to the Thesis	1-10
<b>2 Document Composition</b>	<b>2-1</b>
2.1 Traditional Document Production Techniques	2-1
2.1.1 <i>How do books get produced?</i>	2-2
2.1.2 <i>Roles involved in producing a book</i>	2-6
2.2 The Concept of Style	2-14
2.2.1 <i>Style as a Series of Design Choices</i>	2-15
2.2.2 <i>What Do Styles Affect?</i>	2-17
2.2.3 <i>Styles for Specific Media</i>	2-18
2.3 Early Typesetting Systems	2-19



<b>2.4 Document Compilers</b>	<b>2-22</b>
2.4.1 <i>troff</i>	2-24
2.4.2 <i>Scribe</i>	2-29
2.4.3 <i>TEX</i>	2-30
<b>2.5 Integrated Composition Systems</b>	<b>2-32</b>
2.5.1 <i>Etude</i>	2-34
2.5.2 <i>Janus</i>	2-35
2.5.3 <i>Xerox Star</i>	2-36
2.5.4 <i>Xerox PARC Research</i>	2-37
2.5.5 <i>WYSIWYG – or is it?</i>	2-38
<b>2.6 Document Content Models and Views of Documents</b>	<b>2-39</b>
<b>3 Graphical Style</b>	<b>3-1</b>
<b>3.1 Producing High Quality Illustrations</b>	<b>3-1</b>
3.1.1 <i>Text Book Illustrations</i>	3-2
<b>3.2 Previous Work</b>	<b>3-5</b>
<b>3.3 The TiogaArtwork Prototype</b>	<b>3-6</b>
3.3.1 <i>Tioga Document Model</i>	3-7
3.3.2 <i>Artwork Class Nodes</i>	3-9
3.3.3 <i>Geometric Representation of Illustrations</i>	3-12
3.3.4 <i>Graphical Style Attributes</i>	3-15
3.3.5 <i>TiogaArtwork Rendering Algorithms</i>	3-16
<b>3.4 Results</b>	<b>3-18</b>
<b>4 Tabular Composition</b>	<b>4-1</b>
<b>4.1 What is a table?</b>	<b>4-1</b>
<b>4.2 Early Table Formatting Systems</b>	<b>4-3</b>
<b>4.3 Typographic Requirements for Tables</b>	<b>4-5</b>
4.3.1 <i>Tables are Two-Dimensional</i>	4-5
4.3.2 <i>Typographic Treatment</i>	4-7
4.3.3 <i>Large Tables are Awkward</i>	4-13

## **4.4 Previous Approaches to Table Formatting 4-16**

*4.4.1 The Typewriter Tab Stop Model 4-16*

*4.4.2 tbl Preprocessor 4-17*

*4.4.3 T<sub>E</sub>X 4-19*

*4.4.4 TABLE 4-19*

## **5 A New Framework for Tabular Composition by Computer 5-1**

### **5.1 The Interactive Table Formatting Problem 5-1**

*5.1.1 What do we need to do? 5-1*

*5.1.2 How are we going to do it? 5-3*

### **5.2 The Complexity of Table Formatting 5-3**

*5.2.1 RANDOM PACK 5-4*

*5.2.2 STUB PACK 5-6*

*5.2.3 GRID PACK 5-8*

### **5.3 Table Document Structure 5-9**

*5.3.1 Table Entry Representation 5-9*

*5.3.2 Table Arrangement 5-11*

*5.3.3 Grid Structure 5-13*

*5.3.4 Graphic Arts References to Grid Systems 5-14*

*5.3.5 Style Attributes for Tables 5-16*

*5.3.6 Text within Table Entries 5-19*

### **5.4 Implementation of Table Grid Structure 5-20**

*5.4.1 External Representation of a Table in a Tioga Document 5-20*

*5.4.2 Corner Stitching Data Structure 5-24*

*5.4.3 Overlapping Planes 5-28*

*5.4.4 Grid Algorithms 5-30*

5.5	Table Layout via Constraints	5-35
5.5.1	<i>Constraint Systems</i>	5-35
5.5.2	<i>The Complexity of Linear Constraint Solvers</i>	5-37
5.5.3	<i>The Constraint Table Layout Problem</i>	5-37
5.5.4	<i>Handling Large Tables</i>	5-43
5.5.5	<i>The Layout Algorithm</i>	5-44
5.6	Conclusions	5-45
6	Future Directions	6-1
6.1	Document Model	6-1
6.2	Graphical Style	6-2
6.3	Table Formatter Implementation	6-2
6.4	Table Formatting Algorithms	6-3
6.5	User Interface Design	6-4
6.6	Expert Style Rules	6-5
	Glossary	G-1
	References	R-1
	Index	I-1

---

# Figures

---

## 2 Document Composition 2-1

Figure 2-1. Traditional Graphic Arts Processes 2-3

Figure 2-2. A Hypothetical Publishing Process 2-7

Figure 2-3. Typographic Style Sheet 2-15

Figure 2-4. Cleverness Required to Master T<sub>E</sub>X 2-33

## 3 Graphical Style 3-1

Figure 3-1. Trapezoidal Rule Figure 3-3

Figure 3-2. The Tioga Document Structure 3-8

Figure 3-3. Hierarchical Illustration Structure 3-11

Figure 3-4. Sketch of the Illustration 3-13

Figure 3-5. Geometric Representation 3-14

Figure 3-6. Trapezoidal Rule Slide 3-20

Figure 3-7. Graphical Style Sheets 3-21

## 4 Tabular Composition 4-1

Figure 4-1. The Two-Dimensional Structure of a Table 4-6

Figure 4-2. Vertical Alignment within a Column 4-8

Figure 4-3. Horizontal Alignment within a Row 4-9

Figure 4-4. Transposing a Table 4-14

Figure 4-5. Tab Stops 4-17

## 5 A New Framework for Tabular Composition by Computer 5-1

Figure 5-1. An Approximately Optimal Layout 5-6

Figure 5-2. A Hypothetical Stub Pack 5-7

Figure 5-3. A Wide Range of Content 5-10

Figure 5-4. A Table Entry Box 5-11

Figure 5-5. Hierarchical Tables 5-13

Figure 5-6. Table with Grid Coordinate System 5-14

Figure 5-7. Grid Design 5-15

Figure 5-8. Style Attributes 5-17

Figure 5-9. Alignment on a Character 5-17

Figure 5-10. Horizontal Alignment of Folded Table Entries 5-18

Figure 5-11. Specifying Position within a Column 5-19

Figure 5-12. A Table with a Grid Coordinate System 5-22

Figure 5-13. A Table Object in the External Representation 5-23

Figure 5-14. Corner Stitched Data Structure 5-25

Figure 5-15. Table Mapped Onto A Grid Data Structure 5-26

Figure 5-16. Intersecting Rules 5-29

Figure 5-17. Constraint Variables for a Particular Table Entry 5-38

Figure 5-18. The Constraints Generated for Figure 5-6 5-41

# Acknowledgements

---

This thesis would not have been begun without the two professors who created the Computer Graphics Laboratory at the University of Waterloo and who served as my co-supervisors. When one confronts the decision whether to pursue a PhD, the opportunities to work with and the encouragement from Dr. Kellogg S. Booth and Dr. John C. Beatty (not to be confused with each other!) can not easily be dismissed. The Computer Graphics Laboratory has been a welcome oasis for Canadian computer graphics research and I have enjoyed working there.

The research reported in this thesis was performed at the Xerox Palo Alto Research Center. Xerox has provided exceptional support for this work and has willingly endured my split loyalties between PARC research and PhD research. Maureen Stone and Bill Paxton were responsible for my visiting PARC and later joining the research staff in the Imaging Sciences Laboratory. Several organizational changes caused me to work for Bob Ritchie in the Computer Science Laboratory, where Bob both sheltered me from undue distractions and periodically scrutinized the distractions I was enjoying. PARC has an environment with an incredibly cooperative and collaborative group of computer scientists and the distractions were very tempting. I appreciate my colleagues who patiently waited for my thesis to be finished so that they could work with these tools also.

Maureen Stone and I collaborated on a SIGGRAPH'83 paper that reported the results of the graphical style research presented in Chapter 3. We both share a common interest in high-quality illustrations and her work with Griffin styles evolved independently while I was still in Waterloo.

Bill Paxton implemented Tioga. Tioga is a real jewel in the document composition field and I cannot spend long enough expressing the wonder I felt when I first encountered the Tioga composition system at PARC. It was true pleasure to extend his fine implementation and stretch the boundaries of integrated document composition. Michael Plass designed and implemented the Tioga typesetter and collaborated on the artwork class properties that proved so advantageous in the prototypes I developed. Doug Wyatt crafted the graphics software that serves as the foundation for many of my ideas.

My involvement with the graphic arts and typesetting began in 1974 when Dr. Morven Gentleman convinced the Mathematics Faculty at Waterloo to purchase a small (and cheap) Photon Econosetter. I just could not stay away from experimenting with it when it arrived. Dr. Laurie Rogers, Mark Brader, and Johann George worked on the initial support software; Johann George, Bill Ince, and Alex White collaborated on the initial implementation of TYPESET and its macro packages.

A significant broadening of my appreciation of the graphic arts occurred during 1977 when I collaborated with George Roth, a graphic designer, and John North, a conference chairman, to produce the proceedings for the Third International Conference on Computing in the Humanities (ICCH-3). Over a very short 7-week period, we took 31 conference paper manuscripts, 18 Waterloo co-op students, the Photon Econosetter, and some holiday weekends to produce a hard-bound book in advance of the conference. The manuscripts contained mathematics, computer programs, French, classical Greek, poetry, illustrations, and tables. George Roth very cleverly designed the ICCH-3 proceedings with the considerable discipline required by the limited capabilities of our typesetter and composition software. That initial collaboration grew into a partnership and a company, Waterloo Computer Typography, which subsequently produced several scholarly books and journal articles.

The opportunity of typesetting other people's books became a chance to coauthor an introductory computer science text book. Arnie Dyck, Doug Lawson, and Jim Smith had written a text which I had typeset. When the suggestion arose of my collaborating with them on a revision from WATFIV-S to Pascal, I jumped at the chance. The authors' meetings were very interesting because we had almost total control of the book: graphic design, typography, digital fonts, mathematical typesetting, illustrations, indexing, paste-up. And we were being paid for this learning experience! I am indebted to Arnie Dyck for his attention to detail, insistence on quality, and the great fun we enjoyed teaching together from our book. I marvelled at Doug Lawson for his writing

skills and his tutelage in numerical analysis, which surely helped me pass the PhD comprehensive exam.

Finally, I must acknowledge Beth Beach, who is a typographer in her own right, and who made Waterloo Computer Typography the reliable and quality organization that it is. While we typeset several books together, the deficiencies we suffered with our tools motivated the research presented here. I hope this thesis will help make her job easier, both because the research produced some interesting results and because the thesis is finally finished. Now we can enjoy life together without the late hours and concentration devoted to writing the thesis.



# 1 Introduction

---

## 1.1 Overview

This thesis addresses the problem of formatting complex documents using electronic document composition tools. In particular, the two problems of incorporating high-quality illustrations into documents and of laying out tabular information will be treated in depth. The research reported in this thesis concentrates on techniques and tools for producing electronic documents that will meet the graphic arts standards of quality, both in electronic form and in subsequent hardcopy form. The notion of style applied to text composition systems, separating *form* from *content*, is extended to both graphical illustrations and tables.

Document composition systems have supported the inclusion of scanned illustrations and computer-generated line drawings before, but frequently the results have lacked the quality expected in a typeset document. This thesis presents techniques for specifying illustrations within the document manuscript and for controlling the quality of the resulting artwork by separating form from content in the specification of illustrations.

Document composition systems have also formatted tabular information before, but they have often permitted only limited control over the table formatting parameters and they have imposed restrictions on the possible layouts for tables. This thesis presents a new framework for controlling the table formatting process and for interactively designing a table layout. The table formatting paradigm introduced here can be extended to other two-dimensional

layout problems, such as mathematical notation or complete page makeup.

The notion of *style*, a way of maintaining consistency, runs throughout the thesis. It is the underlying mechanism for managing the complexities of formatting high-quality illustrations and tables. The use of style both simplifies the design choices and enhances the disciplined use of high-quality typesetting.

The thesis includes aids to assist the reader in understanding the typographic subject matter: a glossary of typesetting terms, an index to the thesis, and plentiful diagrams and illustrations. Glossary terms appear italicized in a distinctive *sans-serif* typeface when they first appear in the thesis, such as the term *style* in the preceding paragraph.

Prototype implementations of the document formatting tools were developed in the Cedar programming environment [Teitelman, Cedar] at the Xerox Palo Alto Research Center. The Tioga editor and typesetter were extended to incorporate illustrations and tables using the prototypes. The Tioga document model and style mechanism were extended to accommodate both the graphical and tabular style requirements. The lessons learned from developing and using these prototype systems will be valuable during ongoing research designing a new integrated document composition environment based on Tioga.

This chapter introduces the notion of a document and particular problems in document composition that have led to the research reported in the thesis. The chapter concludes with a brief summary of the material presented in subsequent chapters.

## 1.2 What is a document?

A *document* communicates information both textually and pictorially. An *author* collects, organizes, and presents the information contained in a document. The document may be intended to inform, to persuade, to argue, or simply to entertain. This thesis deals with some of the problems of producing and reproducing documents for effective communication of information.

Historically, documents were handwritten *manuscripts* reproduced laboriously by scribes. Many of these early documents were handsomely illustrated with hand-drawn sketches or images. Early printing processes required hand-carved *printing plates*. The invention of *movable type* enabled people with less skill to prepare printing plates in less time. Movable type also created more freedom to correct and change pages. It is worth noting that the

early success of these printing systems depended on meeting the quality standards set by handwritten and hand-illustrated manuscripts. There has since been a long history of development in the printing process, with a keen regard for quality always at the forefront.

The traditional skills of reproducing modern documents are called the *graphic arts*. These crafts represent a high standard of quality and readability. The artistry of letter forms, the inclusion of fine quality photographs and line drawings, and the uniformity of color when printing many copies of a document are all testaments to the concern for standards in the graphic arts that have evolved over many years. Traditional graphic arts processes are labor intensive and often time consuming. Electronic tools and techniques have already been introduced into these processes, but the standards and skills remain largely traditional. We wish to preserve the quality aspects of the graphic arts, while making the quality more consistent and more accessible through the use of electronic document composition tools.

Authors assemble the text, illustrations, and reference material for a document and then compose a manuscript. The graphic arts staff receives the manuscript and transforms it into a book, report, pamphlet, poster, or newsletter. The author often works with a manuscript that is very different from the final printed result. A great deal of trust must be placed in the graphic artists to produce a document that resembles what the author expects.

The traditional document production process works best when creating a static and long-lived document. The author must plan ahead to guide the reader through this static form of the document. Reading aids, like tables of contents, indices, and cross references between parts of a document, are often necessary to permit different readers to control their individual reading paths through a large document.

The most exciting prospect for introducing electronic composition tools is not in producing traditional documents faster, but rather in producing entirely new kinds of documents more easily. The author working with draft manuscripts should be able to see his information in its final form. The all too common phenomenon of *author's alterations* (changes requested by the author to the typeset *galley*s supposedly in 'final' form) is convincing evidence that authors often do not really comprehend the implications of their manuscripts until they see them in their final form. This is a costly phenomenon that begs to be corrected. Electronic tools can help authors produce better draft manuscripts and thus better (and cheaper) final documents.

Engelbart's vision of electronic tools to augment human intellect, published in 1968 [Engelbart, NLS], contained many elements only now becoming standard in current document preparation systems. His work was based on a timesharing computer with each user (sometimes several participants in an electronic conference) viewing a document on graphic displays and interacting with two-handed input (a five-key handset, a keyboard, and the original mouse pointing device) [Engelbart, Terminals]. Engelbart's on-line document system NLS first introduced structured files. These files represented a document as a hierarchy of information statements with cross reference links among related statements. NLS documents could contain both text and simple line drawing graphics. The reader of an NLS document guided his own exploration of the document by selecting the content of interest, which of the cross references to be followed, the formatting parameters that would apply to the display, the levels of the hierarchy to be made visible, and the amount of each statement to be displayed.

Later, the Hypertext [Carmody, Hypertext] and Xanadu [Nelson, Xanadu] projects evolved similar systems that organized document content in a structured fashion and used graphic displays to present the document.

Electronic documents need not be restricted to presenting static information as are documents printed in hardcopy form. The potential exists for electronic documents to react to the specific reader, perhaps by choosing parts of the document based on the reader's experience or interests, or by connecting a dynamic document to a database and extracting the most recent information available.

Nevertheless, the presentation of information through electronic composition tools must strive to meet the reader's expectations for readability, quality typography, illustration, and organization that have been established by the graphic arts community. The challenge to electronic documents is effectively presenting (in a timely fashion) a wide range of material that includes textual statements (possibly in foreign languages), notation such as mathematical or chemical formulae, tabular presentation of information, photographs, line drawings (possibly with shaded and colored elements), and more.

### 1.3 Personal Reflections on Document Production

The author of this thesis has composed several scholarly books and journal articles through a typesetting company, Waterloo Computer Typography (WATYPE), founded in partnership with a graphic designer. This experience involved the development and application of electronic composition tools for preparing and typesetting scholarly manuscripts for publishers who insisted on traditional graphic arts standards. Despite the benefits of those composition tools, many deficiencies in the tools and a multitude of production difficulties due to a lack of integration among the tools had to be circumvented. Those difficulties have helped to focus the author's current research into illustration and table formatting problems.

A sequence of three introductory texts for computer science, typeset by WATYPE, demonstrates the benefits of storing and editing the manuscript with electronic tools. The first book was based on the WATFIV-S programming language [Dyck, WATFIV-S]. The manuscript was created by the authors using a text formatting processor that could only produce draft copies on a line printer. When the book was contracted for publication, the computer files were translated, using automated text processing tools, into formatting commands for a typesetting system [Beach, Typeset]. Later, two variations of this book were developed for PASCAL [Dyck, PASCAL] and FORTRAN77 [Dyck, FORTRAN77]. In each case, the manuscript files were methodically reviewed and edited to produce the preliminary version of the next book. Much of the material, especially the mathematical presentations and algorithms, remained unchanged and could be used without modification. Completely new sets of computer programs for each programming language were incorporated from the computer files used to compile and test the programs. A robust file system and archiving tools available on the host timesharing system made the job of managing all of these manuscripts and changes practical.

Editorial consistency within a document was achieved more easily using electronic tools than by manual proofreading. Checking words in a foreign language lexicon for a Chaucer bibliography [Peck, Chaucer] or checking the citations of figure captions for heavily illustrated text books were easily accomplished through the facilities of word processing or text editing programs by making global edits. The organization of documents into chapters, sections, paragraphs, tables, figure captions, and reference citations was regulated by defining standard formatting tags or commands within the computer manuscript files. Families of documents, such as the sequence of three text books or all of

the articles in a conference proceedings [Lusignan, ICCH3], shared a common design and layout by using the same composition tools with the same design parameters. With sufficient care and foresight, the resulting documents have significantly greater editorial consistency yet meet the same quality standards of traditional methods.

The accurate presentation of computer programs and computer generated data through electronic composition tools has seen a significant improvement over traditional graphic arts techniques. Manual transcription of data and the misinterpretation of the unusual appearance of computer programs leads to inevitable errors when using traditional methods. WATTYPE contracted to typeset an APL manual [- , APL/66] because the authors refused to proofread the APL notation if it was manually transcribed from their draft manuscript that had been prepared on a computer line printer. Other authors in private conversations have related problems with publishing their programs and data. Typographers are experienced in transforming typewritten manuscripts into typeset books, but not in reproducing the line printer output from computers. *Monospaced fonts*, typical of the fixed pitch characters on computer output devices, are rarely used in traditional typesetting. Typographers often substitute other fonts and treat the material as they would typewritten manuscripts. Authors of computer science texts did not appreciate the unexpected changes made to their programs, which of course invalidated them as computer programs.

One example concerns quoted strings in programs being treated as quotations. Many style books demand that the quotation be set off by matching open- and close-quote marks and further indicate that quotations may have their punctuation moved inside the quote marks and otherwise changed for clarity [van Leunen, Handbook, p 60]. Computer programs require extremely precise placement of punctuation *outside the quotation marks*, and the computer character set often does not distinguish between open- and close-quote marks. Authors who use electronic compositions tools can control the interpretation of unusual material, like computer programs, to ensure the accuracy and consistency of the published form.

One aspect of electronic composition that remains difficult is formatting complex mathematical notation. Tools like `eqn` and `TEX` provide abstract languages for an author to describe mathematical notation. These tools require converting the notation into a complicated syntax, especially complicated when nonstandard mathematical notation is involved. While automated syntax checkers do exist, correcting mathematical notation typically requires typesetting proof copies, marking corrections, and making revisions. There are few

interactive 'what you see is what you get' (WYSIWYG) editing tools for composing mathematical notation.

Few systems understand mathematical concepts to help authors avoid errors. While the lack of understanding permits notational schemes like `eqn` to be quite flexible, accommodating new notation is a major frustration in these systems. Authors frequently invent new notation to serve their purposes, especially in Engineering disciplines, or they make heavy use of notation that is not well supported by the mathematical composition tools. For example, the matrix algebra notation in a sparse matrix text [George & Liu, Sparse Matrices] stretched the capabilities of an `eqn`-like formatter to compose square matrices and align rows across matrix equations. More flexible notation schemes designed to accommodate authors who create new notations are needed. Incorporating support from a symbolic algebra package for checking the mathematical notation, analogous to spelling and diction analysis tools, would provide a better mathematical composition environment.

Most composition systems treat mathematical notation separately from normal text. Thus one cannot freely use mathematical notation in all parts of a document, even though mathematical notation is quite natural in chapter or section headings of technical documents. When the text of the heading in the sparse matrix book was automatically duplicated for use as a *running head*, the running head did not format correctly because the text fonts were different. Similarly, mathematical notation cannot be easily used in figure captions where the size of type is different, in the table of contents where headings have been automatically copied from chapter or section headings, or in index entries where phrases have been automatically collected from throughout the manuscript. A more integrated document content model for objects like text, mathematical notation, and illustrations would help to solve the problem of reusing the same material consistently in different contexts.

Tables of information are also awkward to compose. Each table in the documents composed by WATTYPE staff tended to be treated as a separate design problem, requiring special coding for each one. Table formatting tools are less well developed than text formatting tools, with many special table formatting features not possible or not provided. The content of table entries may be restricted, so that mathematical notation or illustrations may not be acceptable as table entries. Because tables are treated differently than text or mathematical notation, it may be awkward to use the same document style for tables as for the textual parts of a document. Simple tables, especially spreadsheets or tables of computed numeric data, are frequently formatted by special purpose programs,

making it difficult to incorporate such tables within the body of an editable document.

Illustrations remain outside the mainstream of document formatting. The illustration packages currently available either produce results crude by graphic arts standards, or are limited in the range of artwork they produce. Almost all the illustrations for books produced by WATTYPE were drawn by draftsmen at a larger scale and reduced to improve the quality of the reproduction. This led to several difficulties with inconsistent line thicknesses, varying typefaces for labels and captions, and differences among a set of similar drawings.

Formatting large documents into pages is another difficulty with electronic composition tools. Because text books contain hundreds of pages, automated *pagination* techniques are desirable. Unfortunately, in complicated situations, the current algorithms are likely to create unpleasant and unacceptable results, especially in placing figures and footnotes. Each special case has to be handled by manually coding special formatting instructions on how to properly break the page. Of course, each time the document changes, these instructions also have to be changed, and consequently, pagination is left until the very last moment.

A contributing factor to the pagination problem is the cost of formatting an entire document all at once. Some systems are noninteractive and the processing is actually run a batch at a time, typically one batch for each chapter. Multiple runs are needed when the document contains cross references between chapters in order to get the page numbers correct, when parts are automatically numbered in order to get the sequencing correct, and when index entries are automatically collected in order to get the page references correct. These formatting cycles often involve reprocessing a lot of the document that has not changed, thereby wasting resources, increasing costs, and introducing delays.

Another implication, even with interactive systems, is the 'pregnant pause' syndrome, where reproduction-quality output is delayed until the moment when everything has been completely and finally formatted. Though WATTYPE had provided publishers with several drafts, each of which appeared much like the final result, none of the pages could be considered final pages. Publishers gain confidence when they see final pages coming out of the production pipeline. With the pregnant pause syndrome there are no final pages until the last minute. This places considerable faith and stress on the composition system to handle the surge of demand to output a complete document. Several failures in the computer hardware, operating system, storage system, communications system, and typesetter delayed book projects for WATTYPE. Furthermore, last minute touch-ups were always necessary to correct overlooked mistakes or to include



artwork not produced with the system, and these must be handled outside the normal production cycle.

These difficulties, experienced first-hand by the author, lead to a concern for developing incremental and integrated electronic document composition tools. The research reported in this thesis is directed towards an interactive ‘what you see is what you get’ environment that supports a variety of document content and produces the high-quality results expected by graphic arts standards.

## 1.4 The Concept of Document Style

Electronic aids for document production have contributed to the concept of *document style*. A crucial insight is the notion of separating form from content in a document, made explicit in *document compilers* like Scribe [Reid, Scribe thesis] and implicit in many earlier *macro packages* like the `-ms` package [Lesk, `-ms`] for `troff`. Style deals with issues of form: appearance, aesthetics, and understandability of document content. Generally styles have expressed how text is formatted, the typography of text:

“The practice of typography, if it be followed faithfully, is hard work — full of detail, full of petty restrictions, full of drudgery, and not greatly rewarded as men now count rewards. There are times when we need to bring to it all the history and art and feeling that we have, to make it bearable. But in the light of history, and of art, and of knowledge, and of man’s achievement, it is as interesting a work as exists — a broad and humanizing employment which can indeed be followed merely as a trade, but which if perfected into an art, or even broadened into a profession, will perpetually open new horizons to our eyes and new opportunities to our hands.” [Updike, *Printing Types*, quoted in [Williamson, *Book Design*, p 4]]

Electronic composition systems have been more concerned with simple typography [Beach, *Computerized Typesetting*] and less concerned with higher levels of style that apply to nontextual components, such as pages, illustrations and tables [Furuta, *Survey*]. On the surface, it appears that specifying a style is easy.

“To lay down rules of style would be easy enough — we need only consider how things were done yesterday, or how they are done today, or how we prefer to do them ourselves, and to elevate these practices or preferences to the status of dogma.” [Williamson, *Book Design*, p 2]

For most people who prepare documents, many decisions are

institutionalized and thus already made for them. When one must create a document style with the rigorous detail demanded by a document compiler or macro package, the quantity of detail is enormous. Some publishers provide style manuals with hundreds of pages that capture this detail [–, The Chicago Manual of Style, 1982]. Other publishers have felt threatened by revealing the style details to those creating document composition tools [Johnson, JACM style] or they can only provide sufficient detail after several iterations of critiquing samples [Bell, Sc.Am. illustration]. Document style appears to be an area that might benefit from the application of expert systems techniques to capture style rules. For now, because we do not understand very well how or why things are done, we must fall back on replicating how documents were formatted in the past.

## 1.5 Roadmap to the Thesis

The rest of this thesis reviews the state of electronic tools for document composition and details solutions to some of the difficult problems in handling illustrations and tables.

Chapter 2, *Document Composition*, presents a survey of the traditional graphic arts process for producing a document. This includes a review of how books get published and the roles of the people involved in producing a book. Typesetting systems, including early computer typesetting systems, document compilers, and integrated document composition systems are reviewed for their handling of document style, illustrations, and tables. A survey of existing document models highlights the need for more structured models to integrate various kinds of document content.

Chapter 3, *Graphical Style*, extends the style mechanism to illustrations. The same 'form *versus* content' separation so successfully applied to textual objects is applied to graphical objects. A prototype implementation demonstrates the effectiveness of graphical style in achieving this separation and consistency in illustrations. Graphical style is revealed to be insufficient because it does not deal adequately with layout. The observation that specifying positioning constraints within illustrations would help control the layout leads to the consideration of a concentrated layout problem of formatting tables as a constraint satisfaction problem.

Chapter 4, *Tabular Composition*, examines the problems and difficulties in formatting tables. The earliest computer-typesetting programs were for preparing numeric tables but their approaches were simplistic and limited. A survey of the

typographic features required for formatting tables leads to an examination of current table formatting capabilities available in document composition systems.

Chapter 5, *A New Framework for Tabular Composition*, introduces the use of grid systems and mathematical constraint solvers to the table formatting problem. A review of grid systems and their application to table layout provides the basis for incorporating many typographic features into a document structure suitable for tables. The constraint solver provides the general layout engine for formatting tables as well as the basis for an interactive table design tool. A prototype table formatter demonstrates the capabilities for handling complex tables.

Chapter 6, *Future Directions*, discusses several research problems that evolve from the graphical style and table formatting work reported in Chapters 3 and 5.

The *Glossary* explains terms used by typographic specialists. The glossary assumes the reader has a computer science background, and thus does not include common terms from computer science. Terms that appear in the glossary are identified in the thesis by use of a distinctive *italic typeface*.

An extensive list of *References* is provided for more detailed reading about typography, document composition systems, and graphic design.

# 2

# Document Composition

---

This chapter surveys existing techniques for producing documents, beginning with the traditional graphic arts process for turning a manuscript into a finished book. The chapter investigates the concept of document style that arises from the graphic design discipline and which pervades modern electronic composition systems. The use of computers and electronics in document composition is surveyed next, first examining the early typesetting systems, then document compilers such as `troff`, Scribe, and `TEX`, and finally integrated document composition systems such as Etude, Janus, and the Xerox Star. The survey of document composition techniques concludes with a discussion of several issues concerning the structure of information in documents and a description of some models of structured documents.

## 2.1 Traditional Document Production Techniques

Researchers make substantial use of books and journals in their everyday work. However, few people understand how those documents are produced. Only when they decide to write their own book or to edit a scholarly journal do they become involved in the mysterious world of the graphic arts. This survey is intended to help the reader to understand document production, to appreciate the many diverse roles and skills necessary, and to realize the vast number of details and decisions involved in producing high-quality documents.

### 2.1.1 How do books get produced?

An interesting review of how books are produced is contained in the anthology *One Book/Five Ways* [AAUP, *One Book/Five Ways*]. This reports on a comparative publishing experiment in which five university presses prepared the same book for publication: the University of Chicago Press, the MIT Press, the University of North Carolina Press, the University of Texas Press, and the University of Toronto Press.

The procedures used in each press were remarkably common. Although the approaches varied somewhat, all involved the stages of acquisition, market and preliminary cost estimation, editorial revision, design, production, sales, and promotion. Each press documented their procedures, their forms, and the guidelines they applied to the various processes. *One Book/Five Ways* contains a rich collection of raw material for anyone interested in the publishing process.

In particular, the report includes the style guidelines from each of the presses. These guidelines establish the publisher's *house style*, and govern editorial, graphic design, illustration, composition, and typesetting decisions. Perhaps the most well-known style guideline for scholarly documents is *The Chicago Manual of Style*, which was referenced by several presses in this experiment, although most have their own refinements and special instructions.

An important feature of the traditional book production process is the parallelism achieved through several groups working on distinct aspects of a book. When a manuscript arrives at the press for consideration, it is quickly copied and sent out for two or more independent reviews to decide whether to publish the work. Once the decision to publish is made and the completed manuscript arrives from the author, copies are sent simultaneously to (1) the production editor, who establishes a *job docket* to track all of the subsequent stages of the publication, (2) the copy editor, who makes editorial revisions, and (3) the graphic designer, who designs the book and its illustrations. This parallelism is shown in Figure 2-1 for a simplified and hypothetical publication process.

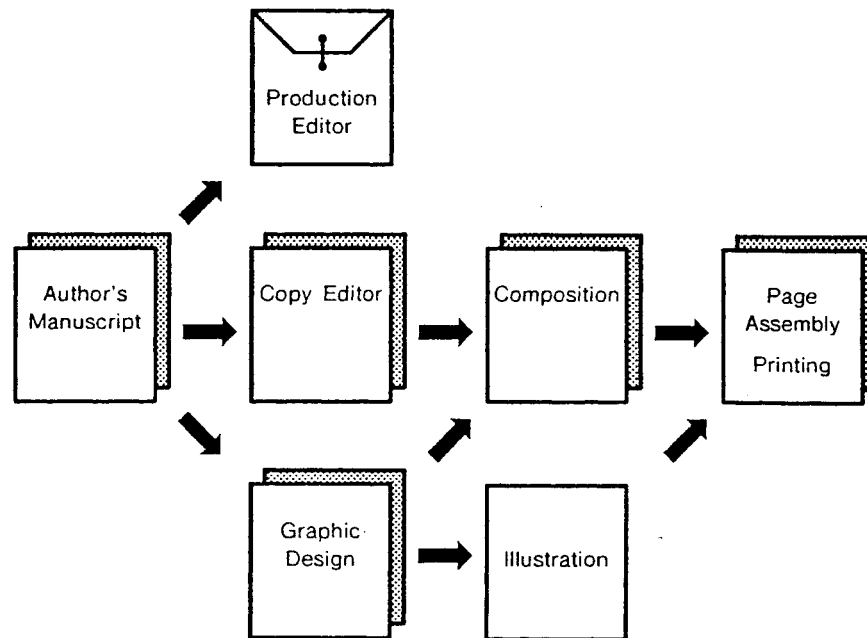


Figure 2-1. TRADITIONAL GRAPHIC ARTS PROCESSES involve considerable parallelism in the procedures for publishing a manuscript. The author's manuscript is copied and sent to the production editor, the copy editor, and the design/illustration department. Edited pages are typeset by the composition staff who are guided by the design of the document. The typeset manuscript and the illustrations are then assembled into pages in preparation for printing.

Other parts of the document publication process also involve parallelism. If the book is to have a *jacket* or cover illustration, that illustration is undertaken while the insides of the book are prepared. The table of contents and Library of Congress submission forms are prepared as soon as the book enters production to ensure that the *imprint page* and the *front matter* of the book are ready for printing.

The index is often on the critical path near the end of the document production cycle. Since index entries must have the correct page numbers, the index can not be fully completed until all of the pages have been assembled. Typically the index entries are compiled in parallel with the book composition. After the page numbers are assigned on the *reproduction pages* (or *page repros*) the index manuscript is completed in parallel with the final proofreading of the book pages.

Even with the use of electronic composition tools, preparation of *back matter* is on the critical path and inconsistent page numbering occasionally results. Such problems appear in the appendices of the second edition of Newman and Sproull's *Principles of Interactive Computer Graphics*

[Newman&Sproull, Computer Graphics], in which the reference citations all refer to a preliminary draft version, because the authors forgot to make 'one last revision pass' over the reference citations in the appendices. The second edition was typeset by the authors using facilities at Xerox PARC because they could complete revisions up to the last minute and control the accuracy of computer programs contained in the text. In a normal production process, there are more people checking things and hence less chance of oversights, such as what actually happened in the appendices.

An area of great concern to the publisher is administration of the production process. Publishers usually have several projects underway at the same time because of the delays involving revisions and approvals from the author of a single project. The *production editor* controls the document publication process for the publisher, determining time and cost estimates for the publication, selecting and contracting with suppliers, tracking the parallel stages of the composition process, and keeping records of deadlines and expenses. In a journal publishing situation, the problem is compounded by the dual pressures of multiple authors and frequent publication deadlines for each issue.

These process control functions are the most important contributions of publishers. Some publishing companies employ little more than production and marketing editors in house, subcontracting most of the skilled jobs such as copy editing, design, illustration, composition, printing. In the electronic publishing or self-publishing process, these subcontracted jobs are performed by the manuscript author and electronic document production tools will have to handle them successfully.

The traditional document production process in the graphic arts routinely accommodates difficult manuscripts. Typically tables, mathematical notation, illustrations, and page layout are aspects of document production that are considered difficult by traditional publishers. The following sections discuss how each one of these areas was handled in the comparative publishing experiment.

### *Tables*

There were only a small number of tables in the *One Book/Five Ways* experiment, but they were always treated separately from the main body of text. Many publishers rely on the skill of the compositor or typesetter to handle tables:

“A good composing room can translate almost any tabular copy in a reasonably clear and presentable example of tabular composition.” [Williamson, *Book Design*, p 160]

*The Chicago Manual of Style* provides authors with the “dos and don’ts” for preparing tables in manuscripts. In particular, authors are expected to prepare tables on separate pages because the tables will be composed separately from the text. There are some cautions also. For instance, the University of Chicago Press no longer prefers vertical rules in tables because Monotype composition (using molten metal casting of individual letters), which could insert a *vertical rule* easily, is no longer economical. With phototypesetter composition, vertical rules are difficult and expensive:

“In line with a nearly universal trend among scholarly and commercial publishers, the University of Chicago Press has given up vertical rules as a standard feature of tables in the books and journals that it publishes. The handwork necessitated by including vertical rules is costly no matter what mode of composition is used, and in the Press’s view the expense of it can no longer be justified by the additional refinement it brings.” [–, *The Chicago Manual of Style*, 1982, p 325-326]

### *Mathematics*

Although there were no mathematics in this experiment, publishers treat mathematical notation very differently than textual material. Kernighan and Cherry note this difficulty in their paper on computer typesetting of mathematics [Kernighan&Cherry, eqn] where they quote the following from *The Chicago Manual of Style*:

“Mathematics is known in the trade as *difficult*, or *penalty copy* because it is slower, more difficult and more expensive to set in type than any other kind of copy normally occurring in books and journals.” [–, *A Manual of Style*, 1969, p 295]

Some publishers specialize in mathematical and scientific documents. They utilize both skilled copy editors and special suppliers to handle the difficult mathematical material. Other North American publishers send mathematics copy to the Far East, where *hot metal* composition provides the quality and cheap labor rates reduce the cost.



### *Illustrations*

The treatment of illustrations varied widely in the publishing experiment described in *One Book/Five Ways*. In one instance, a publisher chose to have an artist prepare line drawings rather than include *halftone* photographs because there were no convenient local suppliers to create *halftone screens* for the photographs. In contrast, another publisher planned photographs for the opening page of each chapter as well as for most of the illustrations. Generally, illustrations are prepared separately while the book is being copy-edited, and are then manually assembled onto the completed pages.

### *Page Layout*

Examining the book design and page layout used by most publishers reveals mainly the results rather than the design process itself. *Page dummies* and *sample pages* are the usual products of the design process. Page dummies are sketches of the page layouts prepared by the graphic designer for approval. Sample pages are pages typeset and assembled by the composition supplier. Both techniques may require several iterations between designer, supplier, and publisher to make certain that the publisher is satisfied and that all the style guidelines are followed. Unfortunately, such an iterative design process generally means that the publisher's guidelines have never been completely specified, frustrating those attempting to become a supplier with new technology.

#### **2.1.2 Roles involved in producing a book**

The document production process is complex. To help understand the process better, this section examines the individual roles of people involved in producing a published document. Anthropomorphism, or the attribution of human behavior to some problem, has proven beneficial in making complex parallel processes more easily understood [Dyment, Corkscrew] [Booth&Gentleman, Anthropomorphism]. An interactive paint program [Beach, Paint] was implemented using multiple processes, where anthropomorphism served to clarify and simplify the relationships of the parallel processes. Through cataloguing the roles involved in document production, the structure of the problem becomes apparent as a set of integrated processes.

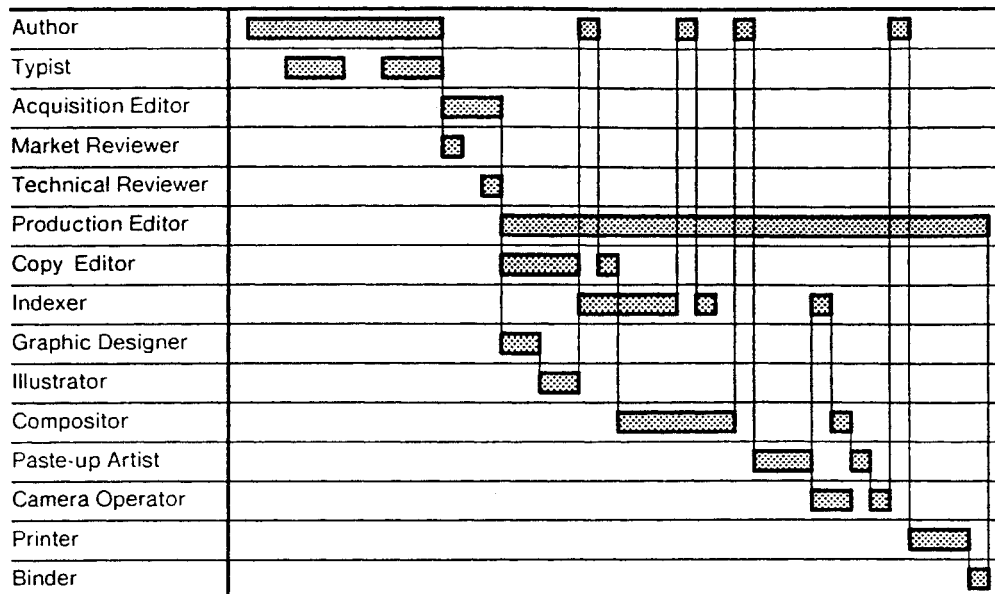


Figure 2-2. A HYPOTHETICAL PUBLISHING PROCESS indicating the roles and their interactions at various stages. The horizontal axis represents elapsed time and the thin vertical lines join activities that begin or end at the same time. Delays or inactivity are not shown, but may exist at many places in the process.

(Aside: An example of the lack of integration in electronic tools occurred when preparing Figure 2-2. There are 15 text labels and the first version of the illustration contained two spelling mistakes. Because the illustration was prepared with a separate illustration tool and was not integrated with the document, the spelling tool used on the text of this chapter was unable to find the mistakes in the illustration.)

An important thing to remember while reading this categorization of roles is that the descriptions relate to activities and not people. Sometimes people may fill several roles at once, such as an author who types and composes the manuscript, or a graphic designer who does the layout, illustration, and paste-up. The use of document composition tools in universities and research labs has tended to encourage (or force) authors to take on multiple roles. From this experience, people may falsely conclude that each job looks easier than it is, especially when they are not aware of what they are doing wrong. Concentrating on each role separately helps us to understand the process and to realize the skills necessary to accomplish all aspects of that specialist's job.

- *Author of the manuscript*

The author creates the original manuscript. Generally, the manuscript is textual material, although for some subject areas there will be vast quantities of mathematical notation, computer programs, tables, line drawings, or photographs. The author may produce several draft manuscripts with the assistance of a *typist*. Some authors now do their own typing with word processors or text editors. Sophisticated editorial tools, such as the diction and writing style analysis tools offered in the UNIX Writer's Workbench [Cherry, Writing Tools] [Macdonald, Writer's Workbench] and in other commercial editing systems [Alexander, Editor Aids], may be used by an author to improve the quality of the writing.

A draft manuscript is submitted by an author to an *acquisition editor* or *journal editor* for consideration. After a favorable publishing decision, the author completes the manuscript and adds front matter that may include a preface, an introduction, acknowledgements, etc. If the document is to be indexed or have other reference material, the author may need to prepare this material also. The completed manuscript is sent to the *production editor*, who begins the publication process. Some publishers will now accept manuscript submission in electronic form, such as word processor diskettes or magnetic tape.

The author may be involved in reviewing decisions made by the publisher. The *copy editor* will mark the manuscript with suggested changes and questions to be dealt with by the author. The *graphic designer* or *illustrator* may send drafts of the book design and illustration artwork for review and approval. There may also be an *indexer* involved, who may send the preliminary index entries to the author for review. The author must also check the composition process by first looking at the galleys and later at proofs of the assembled pages.

- *Typist*

The typist prepares the draft manuscript for the author using a typewriter, word processor, or text editor program. Typewriter composition involves only simple typography, typically with only a small number of type styles. Technical typing with many mathematical symbols is much more difficult and time consuming; some typists resort to hand printing symbols that are unavailable on the typewriter. The layout of typewritten material is free form and requirements are quite relaxed. Tables are easily laid out with fixed-width characters on a typewriter.

The human typist frequently acts as a built-in spelling checker and copy-editing service while transcribing the manuscript.

There are several drafts prepared during the creation of a manuscript. If each draft is retyped to incorporate changes, there is a strong tendency to reduce the number of drafts because of the effort required. Often, the completed manuscript contains partial page inserts pasted or stapled together.

- *Acquisition Editor or Journal Editor*

The acquisitions editor solicits and reviews new manuscripts from authors. Opinions of *reviewers* are sought to determine if the manuscript should be published. The publishing decision is made by a publication board or a committee of journal editors and is concluded by the signing of a publication contract or agreement between the publisher and the author.

- *Reviewer or Referee*

A manuscript reviewer may be asked by a publisher to give one of several opinions. Book publishers refer to these people as reviewers, and journal editors refer to them as *referees*. Reviews made early in the process seek to establish the marketability of a manuscript or the appropriateness of a journal article. Later, more comprehensive reviews seek to assess the subject coverage, research contributions, and technical accuracy of the manuscript. Reviewers are generally most concerned with document content, although in some special cases they may also consider the format or style of a manuscript.

Some reviewers of technical material may use their own typesetting capabilities to capture their comments in the complex notation of the subject area, such as mathematics or computer programming. In some cases, such as computer science journals, the reviews may even be transmitted electronically via electronic mail networks.

- *Production Editor*

The production editor controls the document production process. Initially the production editor deals with the author to ensure that the manuscript has all necessary illustrations, that all the sections of the manuscript are finished, and that permission is obtained to reproduce items from other sources. Copies of the completed manuscript are sent in parallel to the copy editor for editorial revisions and to the graphic designer for book design and illustration.

Production editors contact and select appropriate suppliers for graphic arts services when those services are not available within the publisher.

To help manage and track the various stages of several publications going on simultaneously, the production editor maintains a production database recording the expected services, the date and time each service began and finished, the estimated and actual costs incurred, and the current status of ongoing services. This database exists either on paper as the *job docket* (a large envelope that contains all the partially completed results) or in a computer file.

- *Graphic Designer*

The graphic designer provides the book design and layout guidelines. This design can only be done effectively when the entire manuscript is available, although some designs are attempted with incomplete information and later revised during publication. The design guidelines are written in a *specification sheet* or in a *style sheet* to be sent to the compositor with the copy-edited manuscript (see the example in the next section).

As difficult typographic situations arise, graphic designers may design special guidelines for those not covered in the general scheme, such as designing the layout for tables, and specifying the typography for nested lists of material or for foreign language extracts.

Artwork for the illustrations may or may not be the responsibility of a graphic designer, depending on the designer's agreement, talents, or interests. Jacket or cover designs may also be the graphic designer's responsibility.

- *Copy Editor*

The copy editor ensures that the manuscript meets the publisher's house style for language usage, grammar, spelling, citations, references, illustration captions, table arrangements, headings, lists of items, foreign language phrases, etc. The copy editor deals with all the irksome details that would annoy the reader if they were not treated consistently. For example, the copy editor checks cross references from one section to another for completeness and verifies that captions, footnotes, and citations are numbered sequentially. Missing information or references and questionable corrections are sent to the author for action.

Obviously electronic editing tools greatly assist the copy editor to accomplish these consistency checks. Displaying both the cross reference and its referent through multiple views (or windows) of a manuscript help to check cross

references; pattern-matching search operations permit quick global checks; style and diction analysis tools may be of assistance in checking the grammar, spelling and language usage.

The copy editor marks the manuscript for the compositor by identifying the logical parts of the document, such as chapter openings, various levels of section headings, types of lists of items, and captions for tables and illustrations. Selecting the typographic treatment of those logical parts is the responsibility of the graphic designer, who specifies to the compositor the typography for each part in the style guidelines.

- *Indexer*

The indexer prepares the index entries for a manuscript, assigns page or reference numbers to each entry, sorts them, and creates an index manuscript. The indexing job may or may not be done by the author, although the author usually must approve the index manuscript. The indexer works with the manuscript in two stages: the copy-edited manuscript prior to composition to determine the index entries, and the page proofs to assign the correct page numbers to the sorted index entries. The requirement for correct page numbers places the index on the critical path for publication and some publications omit the index to reduce the delay.

Electronic aids for indexing have not proven to be a panacea. Winograd and Paxton created a general set of indexing tools [Winograd&Paxton, *TEX Indexing*], yet the index still required hand editing and fine tuning. The difficulty in preparing an index is the proper selection and cross referencing of index entry terms or phrases. Skilled indexers still produce better indices than most computer-generated ones because they index on meaning, not on a precise phrase found in the manuscript.

- *Illustrator, Draftsman, Graphic Artist*

The illustrations for a publication are prepared from initial artwork provided by the author. The range of illustrations found in technical documents spans fine hand-drawn illustrations produced by a graphic artist, engineering drawings prepared by a draftsman, and photographs supplied by the author or a photographic service. Often illustrations are produced by tracing the author's sketches, which results in revision cycles as the author more clearly indicates his intentions.

The graphic designer may produce illustration artwork personally or may establish artwork guidelines for original size, reduction factors, line weight, typography, shading textures, materials, and so on. Reducing the original artwork improves the quality of the line drawings by making the line weights appear more consistent (small variations are less noticeable) and by sharpening the contrast in the image. Careful coordination of dimensions and text size on the original artwork is necessary to ensure that the reduced artwork suits the surrounding typography when assembled on the page.

- *Keyboarder, Coder, or Inputter*

The composition of a document is accomplished in two stages: entering the marked-up manuscript into a typesettable file, and then outputting the file on a typesetting device. Typically there is one format code for each logical part of the document marked by the copy editor. For example, there might be a code for the chapter opening, for each level of section heading, for beginning an indented list of items, and for a line of a table. The job of entering the marked-up manuscript may be further subdivided into several phases: assigning *format codes* to the copy editor's marks, designing the typesetter codes for each format, and inputting the manuscript codes and text. The style sheet provided by the graphic designer determines the appearance of marked up parts of the manuscript and hence the typesetter codes required.

The typesettable files may either be entered directly, on less expensive slow typesetting devices, or kept on some storage medium (perhaps paper tape, floppy diskettes, rigid disks or magnetic tape) for more expensive high-speed typesetters. Corrections to the typeset galley proofs are most often made by typesetting corrected pieces of the manuscript, rather than correcting the files and retypesetting the entire galley. In the case of large documents, management of the corrections is a concern and poses difficulties for subsequent uses of the document.

- *Compositor, Typesetter*

The compositor produces the actual typeset output. This person may also do the keyboarding, but a compositor must have the skill to enter specific typographic codes for unusual or difficult typesetting jobs, such as for mathematics, tables, illustration labels, *copy fit* text that must fit certain dimensions, and so on. The compositor runs the typesettable file through the typesetting device and produces the typeset galleys or pages.

- *Paste-up Artist*

Most documents are typeset in galley form and later cut and pasted into page assemblies. The paste-up artist collects all the pieces of the manuscript in their final form: typeset text, running heads with page numbers, mechanical artwork for the illustrations, and photographs. Pages are assembled by cutting apart the galleys into pieces that will fit on each individual page and pasting the pieces onto page layout forms. These layout forms are typically printed with light *blue lines* that will not reproduce on photographic negatives for printing. The paste-up process requires a sharp knife and a waxing machine, which coats the back of photopaper lightly with wax that helps the paper adhere to the layout forms when the two are pressed together. The wax adhesive is pliable so that the pieces can be safely separated if the layout needs to change.

Paste-up only applies to photocomposition systems that produce paper or film original type. With metal *foundry type*, the assembly process involves moving metal type slugs into place and performing craft operations, like surrounding type slugs with *furniture* to provide the spacing for page layout, or  *kerning* individual letter slugs by cutting off the corners to make them fit together better. Some legal organizations have required metal type for legal documents to avoid potential errors in electronic composition systems using phototypesetters [Leith, Metal type]; they wanted to see and verify the final type.

The graphic designer may paste-up a document, especially if the manuscript requires frequent design decisions. In such cases it is quite difficult afterward to determine the rules and logic that were applied to accomplish some of the creative layouts.

- *Process Camera Operator, Stripper*

After the page assembly stage, completed pages are ready for printing. Depending on the printing process, it may be necessary to use a large-format graphic arts process camera to prepare photographic negatives of each page. The negatives are in turn used to expose printing plates. Text and line art illustrations are photographed directly on very high contrast negative film, whereas photographs are screened or *halftoned* to provide the tonal variations on high contrast film. If the printer is capable of printing several pages in one pass, then the stripper must prepare an *imposition* of several pages into one printing *signature*.

The graphic arts process of producing printing plates from assembled pages (*master images*) has been imitated by the concept of rendering



device-independent image masters through page description languages like Interpress from Xerox [–, Interpress] and PostScript from Adobe Systems [–, PostScript].

- *Printer*

The printing process selected by the publisher depends on the number of copies or impressions required. Short-run printing (up to 50 copies) can be printed cost-effectively with a photocopier from a paper original. Medium-run printing (from 50 to 1,000 copies) can be printed with an offset duplicator using an inexpensive paper-based printing plate. Long-run printing (from 1,000 to 10,000 copies) are generally printed with high-speed offset printing presses in *signatures* containing several pages and using metal printing plates.

If the document requires color, then there must be separate impressions made for each printing ink color. Each impression requires a separate master image, one for each color of ink. To print images with a full range of colors, *separations* may be prepared by an outside supplier working from a slide transparency of the colored image. For a small number of flat colors (typically black plus one or two colors) the separations may be made by the process camera operator from color-keyed parts of the original document.

- *Binder*

The printed pages must be collated and bound together to form a completed document. The bindery specializes in taking the bulk pages, possibly in signature form, folding them, collating them in the correct sequence, sewing or otherwise fastening the pages together, and trimming the pages to finished size. The cover, whether a cloth-covered hard-cardboard case or a strong paper back, is attached around the document. Any printing on the cover or jacket must be designed and printed in time for binding. The result is a completed publication ready for distribution.

## 2.2 The Concept of Style

It is important to observe that there are an incredible number of choices in the design parameters that go into producing a document. How do people make the choices? What controls the choices? How are the choices communicated when they are made?

2.2.1 Style as a Series of Design Choices

Many design choices are involved in the process of producing a document. For example, the copy editor chooses names for the logical parts of the document and communicates them to the graphic designer and compositor on the marked-up manuscript. The graphic designer chooses the typographical parameters for these marked parts of the manuscript and communicates them to the compositor on type-specification sheets, such as the one shown in Figure 2-3. The compositor acts on the mark-up codes, using the type specifications, and enters typographical formatting codes in the typesettable file.

Content	Parameters	Typeface			Typesize		Type-weight				Slope		Leading		Lettering			Color		Column-width		Alignment		Indent			
		1.	2.	3.	1.	2.	light	medium	bold	semi-bold	extra-bold	regular	italic	solid	points	all-caps	all-lower	initial-cap	black	pica	justified	unjustified	centred	flush-right	points	indent	
Title	main																										
	subtitle																										
Heading	1st level																										
	2nd level																										
	3rd level																										
	4th level																										
Text	quotes																										
	normal																										
	footnotes																										
	captions																										
Tables	captions																										
Numbering	folios																										
	figures																										
	footnotes																										

Figure 2-3. TYPOGRAPHIC STYLE SHEET typical of the specifications that graphic designers provide compositors to control the parameters of typeset documents.

All of these choices influence the publishing style of the organization. The American Heritage Dictionary's definitions of 'style' and 'style book' help clarify what style means and how it can be used:

“**style** *n.* 1. The way something is said or done, as distinguished from its substance . . . 7. A customary manner of presenting printed material, including usage, punctuation, spelling, typography, and arrangement.” [–, Dictionary]

“**style book** *n.* 1. A book giving rules and examples of usage, punctuation, and typography, used in the preparation of copy for publication.” [–, Dictionary]

Each publishing house develops its own house style, a way of doing things that will distinguish documents from that publisher. In the publishing experiment described in *One Book/Five Ways*, the University of Toronto Press provided the most concise set of composition style guidelines, covering the following topics:

**text composition:** word spacing, word division (hyphenation), letterspacing, paragraphs, leading, small capitals, figures (numerals).

**punctuation:** dashes, periods, apostrophes, colons, semi-colons, exclamations, question marks, ellipses, quotations.

**special settings:** capitals, tables (avoid vertical rules), footnotes, extracts, quotations.

**page makeup:** facing pages, widows.

People at different levels contribute to a publisher's distinctive style. The editorial staff establishes the guidelines for authors and copy editors, such as recommended forms of presentation, spelling, language usage, or the avoidance of vertical rules in tables. Graphic designers select the typography and layout for book designs. The composition staff determines the final typesetting choices through interpreting the typographic specifications.

A publisher's style is developed through an iterative process. The high level plan is established by the publisher and the editorial staff; they request a certain 'look' or 'feel' for a publication. The graphic designer reduces that high level plan into more specific guidelines, but the compositor still has some freedom to interpret typographic choices. The result is sample pages. These pages are passed up the chain for approval and are returned for correction. The changes iterate among publisher, graphic designer, and compositor until the publishing staff finally 'sees' what they want. For large documents, this leads to inconsistencies in how variations not covered in the sample pages are handled, or even differences due to different people working on the manuscript. The solution has traditionally been "Try it again until you get it right."

For automated composition systems that rely on algorithms to carry out repetitive actions, the traditional design process makes it hard to extract the

formatting algorithms from style guidelines. The guidelines are expressed in terms of what people are doing, rather than the process of doing it, or the cause and effect decisions that lead to the result. Therefore, it takes several iterations with sample pages that cover all the expected situations before a creative programmer can express the style rules as an algorithm.

### 2.2.2 What Do Styles Affect?

Style may seem to affect or control more than just the appearance of a document. For instance, consider the choice between Canadian and American spelling, something that might be treated as a style choice. Clearly different spellings contain different letters, as in 'colour' *versus* 'color', 'labelling' *versus* 'labeling', but the same letters may appear in a different order, as in 'centre' *versus* 'center'. The concept of style must accommodate these apparent changes in substance.

We need to realize that style can accomplish changes at many different levels. The change in spelling does not affect the meaning of the sentence containing those words, and therefore the substance of the meaning remains constant while the spelling varies. In fact, many Canadian and American readers easily pass over these different spellings. The style may have changed the characters but not the meaning of the words.

Consider the language processing tricotomy of lexical, syntactic, and semantic analysis. Style can be seen to affect primarily the first two stages of analysis. Style at the *lexical level* affects a token's appearance, such as the choice of spelling. More common lexical style changes are the use of distinctive typefaces for section headings, the inclusion of whitespace above and below section headings, etc. In fact, most typographic parameters fall into this lexical category of style.

Style at the *syntactic level* affects the order of information in the document. One example is the order of names in a bibliographic citation; one style places the surname before initials, while another style places initials before the surname. Another example of syntactic style is the placement rule for parts of a document during page layout, such as locating figures at the top or bottom of a page and collecting all footnotes at the bottom of each column.

Style is also possible at the *semantic level* by providing different readers with different views of the document. For instance, a document on how to use the Cedar mail system on a new kind of file server [van Leunen, One Document] was prepared for readers with different backgrounds. The document contained

written modules of information for one of three kinds of audiences: those who had never used the mail system, those who had used the mail system but stored their files locally, and those who had used the mail system and had some experience with the new file server. A map of which modules applied to which experience categories was used to compile three versions of the document from the various modules. Cargill presents similar ideas for managing different views of software source code [Cargill, Views]. In his scheme, multiple software versions for differently configurable systems were maintained in the same file structure. Depending on the configuration desired, different software versions would be extracted.

### 2.2.3 Styles for Specific Media

Another style dimension is differentiation in media. Traditional printing processes provide some variation in colors and papers, but other reproduction technology and electronic documents span a broader range of possibilities. Documents that become projection slides, posters, or video displays represent some of these.

The notion of device independence in computer graphics can be applied to document formatting. The survey article on document formatting [Furuta, Survey] presents the notion of a 'view' of a document as the device-independent post-processing of a formatted document for a particular device. However, media and device capabilities may influence the appearance and readability of information in a document. In this case, device independence is less desirable. Rather, we wish to reformat the document to take advantage of device characteristics or, put another way, to change the style to suit the medium in which the information will be presented.

Low-resolution devices without color must obviously use different techniques than high-resolution color laser printers. Type families are hard to distinguish on low-resolution devices: 8-point Times Roman on a display screen is difficult to distinguish from any other serif typeface (such as Garamond or Baskerville) because there are so few 'bits' available to display subtle differences. A color image may lose a great deal when viewed in black and white, especially on low-resolution devices that display only a few, if any, grey levels.

## 2.3 Early Typesetting Systems

The early use of computers in graphic arts typesetting systems has been chronicled in several interesting books. One report of a computer composition system is Barnett's *Computer Typesetting* [Barnett, Computer Typesetting], which describes his work at MIT in the early 1960's. Arthur Phillips's compendium *Computer Peripherals and Typesetting* [Phillips, Computer Typesetting] describes the computing and typesetting technologies that were being applied in the graphic arts industry up to the late 1970's. Seybold's classic book, *Fundamentals of Modern Photocomposition* [Seybold, Fundamentals], surveys the first three photocomposition generations and the state of document preparation systems, as well as presenting his seminal thoughts on the problems of area composition (page layout), computer-generated halftones, and integrated system solutions. Phillips's later book, *Handbook of Computer-Aided Composition* [Phillips, Handbook], describes the evolution of electronic tools in the publishing and printing industries. Berg's *Electronic Composition* [Berg, Composition] provides a complete assessment (much in the style of a consultant's report) of the issues in composition systems, the options available, and the pitfalls to be avoided.

Most of the early graphic arts systems used rather small resources and simple approaches to the complex problem of producing typeset documents: Barnett [Barnett, Computer Typesetting] used the IBM 709 at MIT; Seybold [Seybold, Fundamentals] describes composition software run on an IBM 1130; the first stand-alone typesetter at Waterloo, a Photon 737 Econosetter, had only a 4K 12-bit program memory [Beach, PROFF]. These computer programs accepted typographic codes that mimicked the manual actions of a typographer using a hot-metal type-casting machine. The coding structure intermixed action codes with text character codes. Due to the use of *shift-codes*, *super-shift* codes, and even *upper-rail* and *lower-rail* shift codes, the text was often inscrutable for editing purposes.

Table formatting was an early application of computers in typesetting. The earliest such publication, found after an extensive literature search, was the 1962 NBS Monograph 53, *Experimental Transition Probabilities for Spectral Lines of Seventy Elements*, by Corliss and Bozman [Corliss&Bozman, NBS53]. Since computers were generating numeric data and since typesetting equipment was being driven from magnetic tape, it was natural to combine the two together. This monograph contained only a single table and the table formatting was accomplished by a special purpose program. The program is described in more detail in Chapter 4.

Another class of document composition systems evolved from the text formatting programs developed on general purpose computer systems. The evolution of such formatters from Saltzer's RUNOFF document formatter [Saltzer, RUNOFF] is chronicled in Brader's Masters thesis, *An Incremental Text Formatter* [Brader, Incremental Formatter], and later in the *Computing Surveys* article by Furuta *et al.*, "Document Formatting Systems" [Furuta, Survey]. Documents for such formatters were presented as a stream of characters that included embedded control codes. The earliest RUNOFF systems used a period at the beginning of a line of input, an unlikely occurrence in normal written material, to indicate the presence of a formatting command. Later systems escaped from the 'line of input per command' restriction by designating command delimiters as infrequently used characters like braces [Beach, Typeset], backslashes [Ossanna, troff] or at-signs [-, SCRIBE]. Macro and conditional execution facilities for commands extend the range of document formatting possibilities. One tenet of documentation folklore at that time was that if you could make writing a document more like programming, then programmers would take the time to prepare documentation for their work, something which proved difficult to ensure. Unfortunately this was the wrong paradigm. It did not make writing a document easy and it did not get programmers to write better documentation.

The model of a document as a stream of text with embedded commands survives today as a prevalent document formatting model. One consequence of the stream document model in both the early graphic arts systems and the early document formatters is the need to accept the document stream as an abstraction of the formatted document. An early system by Engelbart provided an alternative document model and several alternative views of the document.

The editing and formatting part of Engelbart's augmented human intellect system, NLS [Engelbart, NLS], provided a concrete view of formatted documents as they would appear when printed, without the intrusion of formatting commands. The NLS system was the original 'what you see is what you get' document formatting system and Engelbart coined the phrase *WYSIWYG* (pronounced whizy-wig) to describe it. Due to the limitations of the display and printing devices, NLS was exactly a WYSIWYG system. Many later systems also claim to be WYSIWYG, but cannot claim to render printed output exactly on the display, mainly because of differences in fonts and character widths between the display and the printing devices.

In a further departure from the stream of text and embedded commands model, the NLS system represented the document contents in a tree-structured hierarchy of text blocks, such as the common hierarchy of chapters, sections,

subsections, and paragraphs. A reader of the on-line document could display one of several views. For example, one viewing parameter controlled whether the structure labelling was visible or not, another parameter controlled the number of hierarchy levels displayed, and yet another controlled the number of lines displayed in each text block. NLS could also incorporate line drawings within documents by allowing a graphical object to take the place of a paragraph.

Sadly, these ideas were not widely accepted at the time when they were first introduced in the late 1960's. Almost a generation passed before Engelbart began to receive the appropriate credit for the ideas of the mouse pointing device, multiple windows, and WYSIWYG formatting systems.

Many early graphic arts typesetting systems did not attempt to deal with page layout but only produced typeset galleys to be pasted-up manually in the normal way. The RUNOFF-style formatters provided some limited page breaking capabilities and they could print running heads and footnotes. Such formatters relied on the simple and easily-handled dimensions of fixed-width characters on a line printer or teletype page to make the algorithms workable.

Typesetting document formatters based on extensions to the RUNOFF model could produce output for typesetters. They produced typeset pages by executing page breaking algorithms coded as macros. Some early typesetting work with PROFF [Beach, PROFF], a RUNOFF-like formatter for the University of Waterloo's Photon Econosetter, used simple page depth measurements to break large documents into pages. This was done mainly to avoid the manual paste-up stage due to a lack of available manpower to handle the number of pages produced. Seybold [Seybold, Fundamentals] outlines many of the concerns and difficulties with page layout or area composition addressed by commercial typesetting suppliers.

More complex typesetting systems for high speed typesetters, like the Page-1 composition language [Pierson, PAGE-1] for the RCA Videocomp, permitted more complex page breaking logic to utilize the typesetter for very large documents better. Page-1 is one of the few early composition systems with widely available published documentation. A programmer could write a page breaking algorithm and style handling routines in the Page-1 language, have that compiled, and then execute the resulting composition program against the document input data.



## 2.4 Document Compilers

A significant stage in the evolution of document formatters occurred when the embedded formatting commands in the document began to describe the logical content of the document. These logical commands, or formatting *tags*, require an additional level of indirection to associate the detailed formatting attributes with each tag. Initially this association was provided by a macro processor. Each tag was treated as a macro name. Expanding the macro produced the primitive formatting commands necessary to format that part of the document.

For example, with logical commands one could specify that a part of a document was a heading. By including a tag like `.heading`, one could replace a sequence of detailed commands like “leave 24 points of whitespace, select Times Roman bold typeface, use 14 point type size, and produce unjustified line endings.”

Later systems like Reid's Scribe introduced the notion of compiling a document [Reid, Scribe thesis]. The tags in a Scribe document identify the document parts that are compiled using a *tag definition* database to supply the formatting attributes for a suite of built-in formatting algorithms.

Since more processing is required to interpret macros or compile a document, the development of document compilers was restricted to large general purpose computer systems, typically in universities and industrial research laboratories. Most commercial graphic arts systems remained on less expensive and smaller mini-computers and chose not to provide these 'more expensive' features.

This introduction of document compilers coincides with increasing support for document style. The indirection from tags to detailed formatting instructions emulates the style sheet concept used by graphic designers. Designing tag macros or formatting databases is separated from the marking up of a manuscript. A document style can be shared among a set of documents, for example, among the chapters of a book, the theses written at a university, or journal articles submitted to a particular journal. With such tools, authors who lack the skills for document design can still produce good-looking documents by choosing a document style database and inserting the appropriate tags within their document.

The separation of document *design* and *markup* enables the document content to be reused in different situations by changing the style definitions

associated with the formatting tags, without changing the manuscript or the tags themselves. At Bell Laboratories, where `troff` was developed, a manuscript could be published in three forms: first as an internal memorandum circulated within the lab, second as a technical report cleared for external review, and finally as a published journal article. A single set of tags within the document sufficed by substituting different style parameters for each of the three forms.

The notion of compiling a document implies a massive undertaking. Indeed, problems with compiling monolithic documents occur frequently. Large documents often evolve from smaller ones rather than being planned, requiring more computing resources to format, longer turnaround time, and introducing longer delays in producing drafts of the document. There is a constant tension between the simplicity of making the document out of smaller modules and the complexity of managing the pieces. Simple problems like numbering pages sequentially between pieces can be a problem with some document compilers.

Document compilers exhibit similar debugging problems to those found in compilers for programming languages. An example of a bug in a compiled document is the production of fifty typeset pages with a column width of 1.5 inches because the logic of a macro failed to reset a temporary change in line width. Debugging tools for document compilers have modeled program debuggers such as syntax checkers, simulators of the final output device on less expensive or faster devices, and interactive previewers to display the typeset document on a graphics display. The complexity of writing document format designs in the language of the document compiler leads to the need for 'gurus,' 'experts,' and 'wizards,' just as for complex programming languages.

Certainly, document compilers make some kinds of changes much easier. For example, correcting a chapter heading in one place can automatically affect the chapter opening, the table of contents, and the running heads for that chapter.

Some aspects of documents may not be handled very well or at all by a particular compiler. Difficult composition features are sometimes left for future development, such as mathematical and tabular composition, the incorporation of line drawings and scanned images, or complex page layout designs. The inability to integrate all aspects of the document leads to special handling of the unintegrated parts of the publication, resulting in pasting up artwork for illustrations or special notation typeset separately. Other typographic problems may require specific commands to override the automatic compiled algorithms, such as forcing page breaks to avoid one-line widows, and inserting explicit line breaks to avoid rivers of whitespace or awkward hyphenation problems. Final

corrections and revisions are frequently done by manual cut and paste methods because recompiling the corrected document would take too long or would create new problems, especially with page breaks. Of course, various document compilers do better than others with these problems.

The following sections describe aspects of three document compilers in widespread use, `troff` on UNIX, the Scribe portable document compiler, and Knuth's `TEX`. Of special interest will be the way these systems handle document style, mathematics composition, illustrations, table formatting, and page layout. The survey articles mentioned above [Brader, Incremental Formatter] [Furuta, Survey] discuss additional aspects of document compilers.

### 2.4.1 `troff`

The `troff` document formatting language developed by Ossanna [Ossanna, `troff`] and distributed for UNIX systems is perhaps the most widely used document compiler. The earliest UNIX application was preparing patent applications with `troff` [Ritchie, Turing Lecture, p 758]. `troff` encompasses a family of document compilers. All accept the same formatting commands but differ in their formatting algorithms, which are sensitive to output device characteristics: `nroff` formats for typewriter and line-printer devices with fixed width characters; `troff` formats for typesetting devices with multiple fonts and variable width characters. Porting `troff` to other typesetting devices was very difficult. An output device independent version, `ditroff` [Kernighan, `ditroff`], was created by Kernighan to handle a wide variety of typesetting devices and laser printers, although the formatting algorithms were essentially unchanged from `troff`.

The `troff` formatting language has remained essentially constant since the late 1970's. There are primitive functions for controlling the formatting algorithms and the output device, establishing parameter values, selecting type fonts and sizes, positioning characters, and drawing lines. Additional primitives provide programming support for writing macros and building data structures, such as strings and diversions of formatted text. The command name space is severely limited to two-character tags. Generally lower case letter tags are reserved by convention for `troff` primitive commands and combinations of upper case letters and graphic symbols are used for macro commands. Commands are embedded in the document, either occupying an entire line of input beginning with a command character, or included within lines of input delimited by a backslash character.

The strength of the `troff` document formatting system is the collection of tools implemented as preprocessors. These preprocessors include `tbl` for formatting tables [Lesk, `tbl`], `eqn` for typesetting mathematics notation [Kernighan&Cherry, `eqn`], `pic` [Kernighan, `pic`] and `ideal` [van Wyk, `ideal`] for drawing illustrations, and `refer` [Lesk, `refer`] for producing bibliographic references.

The filter/pipe model from UNIX has determined the architecture of the `troff` document formatting system. The filter model forces the document file to be a linear stream of characters. Each tool reads the entire document file and produces a modified version for the next tool in the pipeline. The recommended processing order is `refer`, `pic`, `tbl`, `eqn`, then `troff`, a convenient order for the majority of documents. Occasionally, when it is not possible to establish a sequential processing order, this scheme breaks down and elaborate techniques to break circular dependencies are needed. Otherwise, the material cannot be formatted by `troff`. Nonetheless, collecting several types of diverse content in a complete document manuscript is more convenient for the author than managing the separate pieces.

Each tool distinguishes its commands in some unique way. For instance, `eqn` processes embedded mathematical notation by recognizing its own delimiters different from other formatting commands. This leads to a hiding of information among various tools. For example, the spelling checker does not investigate any misspelled words inside `eqn` or `tbl` commands even if they are English phrases. Some document commands are treated differently at different stages in the pipeline. For example, `.TS` and `.EQ` are `tbl` and `eqn` commands respectively to begin formatting tables and displayed equations. Later, these commands are passed on to `troff`, which treats them as ordinary macro commands to layout a particular table or displayed equation.

An unfortunate consequence of executing the macro processor last in the `troff` pipeline is the preclusion of style facilities or indirect definitions of formats for tables, mathematical notation, illustrations, or any other preprocessor to `troff`. Some preprocessors furnish their own simple and different macro languages while some users invoke their own preprocessor to provide the missing macro facilities.

Yet the unifying concept of the pipe mechanism provides the `troff` document formatting system with its simplicity. Should the need arise, it is easy to create your own tools to solve difficult document content problems. The ubiquitous document model of a stream of characters with embedded commands makes this possible.

Document style in `troff` is provided by its macro packages. Two frequently used packages are the `-ms` and `-me` packages, the former created by Mike Lesk at Bell Labs [Lesk, -ms] and the latter by Eric Allman at UC Berkeley [Allman, -me]. Macro packages provide two alternative techniques for defining different document styles: one can either parameterize the behavior of the macros or replace the macro package with another that defines the same commands with different effects. As an example, the Bell Labs `-ms` package can format title pages in different ways by initializing parameters from the `.RP` command (released paper format) rather than `.TM` command (Bell Labs technical memorandum format). Also, several variants of the `-ms` package exist for formatting documents in styles suitable for the *Journal of the ACM*, *Communications of the ACM*, and ACM conference papers [Johnson, CACM].

Mathematics composition is provided within `troff` by the `eqn` preprocessor. This mathematics typesetting system has become widely emulated and variations have appeared at other research centers [Gruhn, YFL] and in commercial typesetting systems [Alexander, Micros]. The basic technique is to define a notation language that expresses various two-dimensional relationships among boxes. These relationships may affect the size of boxes, such as making brackets larger around large fractions, or their relative arrangement, such as positioning superscripts and subscripts. `eqn` knows nothing about mathematical concepts or the actual dimensions of the boxes. It relies on the author to provide the precise spacing or line breaking of mathematical notation, and on `troff` to do the actual positioning and formatting of the boxes. Unfortunately, `eqn` guesses about some size relationships and it must be told the current type size explicitly. `eqn` provides built-in relationships for common mathematical notation, but the set of notations is not extensible. However, the `eqn` macro facility (separate from `troff`) and some low-level positioning primitives do provide an escape mechanism for creating new notation as macros. As there are different versions of `troff` for different device classes, there are also two versions of the mathematical typesetting system: `neqn` for typewriter devices and `eqn` for typesetting.

This lack of knowledge of mathematical concepts in `eqn` is both a strength and a weakness. Without any knowledge in the mathematical formatter, one is forced to supply in tedious detail all the necessary spacing for operators. On the other hand, the absence of built-in knowledge avoids having to circumvent inadequate rules when they must be broken.

The two illustration preprocessors, `pic` and `ideal`, provide elementary facilities for including line drawings within documents. The two differ in the

mechanisms for defining the line drawings: `pic` uses a line and curve paradigm while `ideal` uses nonlinear constraints to define boundaries and connected lines. The illustration tools have only rudimentary style facilities for solid and dashed lines and for arrow heads. More elaborate styles, such as various line weights, fancier arrows, textures, and shadows are not provided. Through the preprocessor architecture, and subject to the pipeline order of preprocessors, it is possible to include any `troff` material in the illustrations, including equations and formatted text.

The `tbl` table formatter is a very comprehensive facility capable of formatting almost any table design. Evidence of its power is shown by one author who created boxed illustrations for his paper with `tbl` when a line drawing tool was unavailable [Rosenthal, Graphical Resources].

Even with its flexibility and generality for table formatting, it is awkward to achieve consistent table styles in `tbl`. The user of `tbl` must carefully specify all the layout parameters in a consistent fashion. There are no separate macro facilities within `tbl` to help, and the `troff` macro processor executes later in the pipeline, after `tbl` has processed all of the table information. There is no interactive design tool for tables to assist with the specification. An extensive search for such tools found only a prototype built by Biggerstaff as part of an experiment in object-oriented program design [Biggerstaff, TABLE], described more fully in Chapter 4.

The page layout mechanisms in `troff` depend on two powerful ideas: traps and diversions [Witten, Traps]. A trap is a macro to be executed at a measured distance from the most recent page break in the output stream. The trap macro might, for example, emit the footnotes at the bottom of a page of text. A diversion is an alternate output stream, distinct from the normal stream which is directed to the output device or file. Internal diversions are used to capture formatted information, such as footnotes, for later inclusion on a page. Floating a table or illustration from where it occurs in the manuscript to where it will next fit on a page is accomplished with diversions in `troff`. When a table is first encountered, a new diversion is started to capture the formatted table. After the table has been formatted, the diversion is closed and the macro package can check for sufficient space on the current page to hold the diverted table. If there is room then the diversion is copied immediately onto the normal stream, otherwise it is held until the beginning of the next page. (Complications arise if the table is larger than the page but they need not be considered here.)

Implementation restrictions within `troff` have persisted for a decade. `troff` is an old program, originally coded in assembly language for the DEC

PDP-11, subsequently translated into C. It has changed little since. The facelift for device independent `troff` concentrated mainly on font data structures and generalizing the output device model. The most notorious restriction is the two-character name limitation. Macro packages for `troff` have used elaborate conventions to avoid catastrophe due to name conflicts. Each preprocessor requires some set of macro and register names to support its operations and therefore each reserves some set of two-character names. Users of `troff` must tread gently when creating their own macros or extending the existing packages to avoid name conflicts because the name space is so limited. Limited internal data structures are another annoyance, restricting the complexity of lines of text, the number of columns in a table, or how many entries may exist within a matrix.

The author of this thesis led the development of a document compiler, TYPESET [Beach, Typeset], to alleviate many of the shortcomings of `troff`. A small typesetting business, WAITYPE, used TYPESET extensively to create technical and scholarly publications for a variety of publishers, mainly in mathematics and computer science. TYPESET was built with full knowledge of the `troff` system and attempted to eliminate many of its limitations. The macro processor has a similar syntax and flavor to GPM [Strachey, GPM]. Conditional execution and control structures were added to the macro language, providing document layout programmers with more freedom in expressing their designs. Register names of any length were stored in a hashed symbol table. Most data structures were dynamic and could grow as necessary. Math typesetting was based on the `eqn` design [Kernighan&Cherry, eqn] but the implementation incorporated many enhancements to improve the quality of formatted equations and to facilitate more options for aligning equations and handling matrices. A table formatting package was built using the macro language and provided several additional typographic facilities suitable for style control over the table design. Pagination and layout algorithms were based on trap and diversion paradigms similar to Page-1 [Pierson, PAGE-1] and `troff`. TYPESET was not fully exploited because it required significant programming skill to design new documents and it lacked sufficient documentation. Nonetheless, it did serve well to express difficult book designs and produce competitive graphic arts quality typesetting. TYPESET serves as an interesting contrast in goals with Scribe, described next.

`troff` is in widespread use and continues to have a significant impact on technical document production. Its strengths are the simple document model and the large number of preprocessors and tools that can manipulate documents.

It provides the broadest range of functional capabilities for mathematics, tables, and illustrations. There are several difficulties that limit its effectiveness: implementation limitations, expensive and resource intensive computation, and restrictions on the inclusion of mathematics, illustrations, and tables.

### 2.4.2 Scribe

The second document formatting system in widespread use is Scribe [Reid, Scribe thesis], which was the first to use the term 'document compiler.' The goal for Scribe was to format documents in a portable fashion across document styles, across output devices, and across various computer system installations. Scribe is widely used among the ARPANET community and is distributed commercially [-, SCRIBE].

The notion of the *form* of a document, or its style, as opposed to the *content* of a document or marked up manuscript was made explicitly separate in Scribe. Style information is maintained in a database under the control of a database administrator. The database contains various formatting environments, each specifying a vast number of formatting attributes. Normal users of the document compiler cannot create new environments or attributes. The compilation process takes in a marked up manuscript file and creates a formatted document file suitable for printing.

Unlike *troff* with its macro packages, Scribe provides only built-in formatting algorithms that are parameterized by the environment attributes. The Scribe document formatting language is declarative only. Reid defends the absence of procedural facilities in the database language because 1) a procedural language would reduce the feedback from users when they were unable to do something in Scribe, and 2) without enforcement or user training, "programmability invariably leads to a diversity of style" [Reid, Scribe thesis, p108-109]. Reid concedes that an algorithmic language would increase the usability of the compiler (the goal of this author's TYPESET system). Reid concludes that:

"Furthermore, a programmed system implemented by a diverse variety of people without central control, namely the union of the procedural extensions with the basic system, will invariably be more obtuse and difficult to understand and use than a unified one." [Reid, Scribe thesis, p109]

The Scribe system provides many services for document writers and Reid coined the phrase 'writer's workbench' [Reid, Scribe thesis, p71] to describe the collection. Included among these facilities are the automatic collection of entries



for tables of contents, indices, and glossaries, the automatic cross referencing within a document through symbolic labels, the collection and sorting of index entries, the management of large documents composed of many component files, and the extraction of bibliographic citations from a database of reference entries.

The lack of an algorithmic formatting language has prevented the proliferation of special purpose formatting preprocessors like those for `troff`. There are some mathematical formatting capabilities, but they are of limited capacity, sufficient for some technical documentation but limited for more concentrated mathematical documents. The lack of recursion in Scribe has been a serious impediment to building a mathematical formatter; overprinting is the only readily available technique that can handle the recursive nature of mathematical expressions [Monier, Scribe math].

Table formatting in Scribe is simple to use, but again limited in functionality. The scheme is based on extending the notion of typewriter tab stops that define column boundaries. Scribe provides several typographic capabilities, such as centering within tab stops and filling with leaders, but more general facilities such as centering headings over several columns requires changes to the tab stop settings.

Scribe does provide a nonportable capability for scanned illustrations. Image files scanned for a particular class of output device may be incorporated into a document. Scribe will manage the whitespace layout described for the image, but expects the device to output the image file.

The major accomplishment of Scribe was its successful separation of form from content in a document. The database of formatting environments takes advantage of the special skills of document designers and shares the design among document creators. The range of document content beyond text is limited, and there are few options for building special purpose formatters or preprocessors due to the lack of a formatting language. The next document compiler deals directly with formatting algorithms.

### 2.4.3 T<sub>E</sub>X

Donald Knuth has made document formatting a legitimate topic for study in computer science by his work on T<sub>E</sub>X [Knuth, The T<sub>E</sub>Xbook]. The boxes-and-glue model serves as the basis for algorithmic research into document formatting. Three algorithms have resulted from this work: optimal line breaking [Knuth, Line Breaking], hyphenation [Liang, Hyphenation], and optimal page breaking [Plass, pagination]. The T<sub>E</sub>X document compiler incorporates this model and these

algorithms to provide a comprehensive document formatting system that extends beyond text to mathematics, tabular matter and complex typography.

Typesetting mathematics was a primary goal of Knuth's work on T<sub>E</sub>X [Knuth, AMS lecture]. The notion of composing mathematical expressions from boxes surrounding each character and composing equation boxes for the arrangement of other boxes applies directly to mathematical notation. T<sub>E</sub>X relies on both a large font library to represent mathematical symbols, and a set of positioning operators. The T<sub>E</sub>X typesetting language is a linear expression of the boxes-and-glue model for formatting two-dimensional notation.

Related work on METAFONT [Knuth, METAFONT] resulted in a font design tool capable of producing the many mathematical symbols and alphabets used in T<sub>E</sub>X. METAFONT relies on linear optimization and equation solvers to determine the outline shape of character designs specified by small METAFONT programs.

T<sub>E</sub>X provides a macro definition capability which permits the introduction of shorthand inclusion of complex formatting commands and repeated text. There have been a small number of macro packages developed for T<sub>E</sub>X. Perhaps the most widely distributed is Lamport's L<sup>a</sup>T<sub>E</sub>X package [Lamport, L<sup>a</sup>T<sub>E</sub>X].

Document layout is expressed through implicit controls in T<sub>E</sub>X. T<sub>E</sub>X uses one global algorithm for many layout situations. Thus when breaking lines, there is no explicit notion of centering. Instead one surrounds centered text with two gobs (a technical term in T<sub>E</sub>X) of glue with large but equal stretchiness values. The justification (glue-setting) algorithm fixes the glue size to accommodate all the boxes within the given line measure. Similarly, page justification algorithms are influenced indirectly by gobs of glue between lines of text or parts of a page to accomplish the vertical layout.

Plass's work with dynamic programming optimization algorithms lead to the development of line- and page-breaking algorithms for T<sub>E</sub>X [Knuth, Line Breaking] [Plass, pagination]. The optimization goal is to minimize some badness criteria, such as the sum of penalties for breaking a line in some way. Examples of line-breaking penalties are inserting a hyphen between syllables in a word, hyphenating very short syllables, and introducing hyphens in two or more successive lines in a paragraph. Given the set of boxes and the set of penalties, the optimization algorithm determines the optimal break points. The line-breaking algorithm is part of the current T<sub>E</sub>X82 release, but the page-breaking algorithm is not because it is too resource-intensive and/or too slow.

Sometimes the algorithms in T<sub>E</sub>X produce beautiful results, but they require very clever designers. In this regard, the following summary comment appeared in the *Seybold Report on Publishing Systems* when discussing Tyxset, the first implementation of T<sub>E</sub>X available commercially on a microcomputer:

‘There are, however, some serious flaws. The greatest of these is the need for access to Xenix and T<sub>E</sub>X ‘gurus.’ This would be necessary, we think, for all but the most trivial work.” [Alexander, Tyxset, p 14]

Many situations that can be handled by T<sub>E</sub>X are collected into the ‘Dirty Tricks’ appendix of *The T<sub>E</sub>Xbook*. One example of both the power of T<sub>E</sub>X and the excessive cleverness required to master T<sub>E</sub>X is the inclusion of leaders in an index entry [Knuth, *The T<sub>E</sub>Xbook*, p 392-394]. The example in Figure 2-4 provides the T<sub>E</sub>X codes needed to format the given input for various line measures.

Tables can be handled within T<sub>E</sub>X. However, T<sub>E</sub>X tables rely on horizontal and vertical justification primitives that align in one direction or the other but not both simultaneously. *The T<sub>E</sub>Xbook* demonstrates the ability of T<sub>E</sub>X to reproduce some of the sample tables from the `tbl` manual as evidence of the functionality of T<sub>E</sub>X.

T<sub>E</sub>X is valuable for the algorithmic foundations it brings to document formatting. The interface to those algorithms remains a stream document model without any structure. No WYSIWYG interactive composition system is yet based on T<sub>E</sub>X.

## 2.5 Integrated Composition Systems

An integrated document composition system provides a more direct way of working with documents. One aspect of integration is the combining of the editing and formatting tasks. Document compilers require one to first create or edit a separate manuscript file, then to ask the compiler to turn it into a formatted document. In an integrated system, changes to the document become visible as they are made. Another aspect of integration is the integration of a variety of document content beyond simple text, such as line drawings, scanned images, mathematics and tables.

The first integrated composition system was Engelbart’s NLS, developed during the late 1960’s. The NLS system introduced the notion of ‘what you see is what you get,’ presented a visual interface to accelerate human understanding of the document, accepted direct manipulation of the document structure and appearance, separated the form of the document from its content, and integrated many abstract objects into a uniform representation.

---

ACM Symposium on Theory of Computing, Eighth Annual  
(Hershey, PA., 1976) .....

1879, 4813, 5414, 6918, 6936, 6937, 6946, 6951, 6970,  
7619, 9605, 10148, 11676, 11687, 11692, 11710, 13869

ACM Symposium on Theory  
of Computing, Eighth  
Annual (Hershey, PA.,  
1976) ..... 1879, 4813, 5414,  
6918, 6936, 6937, 6946, 6951,  
6970, 7619, 9605, 10148, 11676,  
11687, 11692, 11710, 13869

ACM Symposium on  
Theory of Computing,  
Eighth Annual  
(Hershey, PA., 1976)  
..... 1879, 4813, 5414,  
6918, 6936, 6937, 6946,  
6951, 6970, 7619, 9605,  
10148, 11676, 11687,  
11692, 11710, 13869

```

\hyphenpenalty10000 \xhyphenpenalty10000 \pretolerance10000 % no hyphens
\newbox\dotbox \setbox\dotbox=\hbox to .4cm{\hss.\hss} % dot box for leaders
\newskip\rrskipb \rrskipb=0.5em plus3em % ragged right space before break
\newskip\rrskipa \rrskipa=-0.17em plus-3em % ragged right space after break
\newskip\rlskipa \rlskipa=0pt plus3em % ragged left space after break
\newskip\rlskipb \rlskipb=0.33em plus-3em % ragged left space before break
\newskip\lskip \lskip=3.3\wd\dotbox plus1fil minus0.3\wd\dotbox % for leaders
\newskip\lskipa \lskipa=-2.67em plus-3em minus0.11em % after leaders
\mathchardef\rlpen=1000 \mathchardef\leadpen=600 % constants used
\def\rrspace{\nobreak\hskip\rrskipb\penalty0\hskip\rrskipa}
\def\rlspace{\penalty\rlpen\hskip\rlskipb\vadjust{}}\nobreak\hskip\rlskipa}
\uppercase{
\def\~{\nobreak\hskip\rrskipb \penalty\leadpen \hskip\rrskipa
\vadjust{}}\nobreak\leaders\copy\dotbox\hskip\lskip
\kern3em \penalty\leadpen \hskip\lskipa
\vadjust{}}\nobreak\hskip\rlskipa \let\~=\rlspace}
\everypar{\hangindent=1.5em \hangafter=1 \let\~=\rrspace}}
\uppercase{\~ = 0 \parindent=0pt \parfillskip=0pt

```

Figure 2-4. CLEVERNESS REQUIRED TO MASTER T<sub>E</sub>X is exemplified by one of the 'dirty tricks' from *The T<sub>E</sub>Xbook*, page 392-394. The top three examples show the same index entry formatted with different line lengths. Note that the dots behave differently depending on whether the two parts fit on the same line or not. The T<sub>E</sub>X code fragment is shown at the bottom.

---

In the following survey, several more recent integrated document composition systems are reviewed. The first two, Etude and Janus, are research projects and deal mainly with document structure and interaction issues. The Xerox Star system tackled the integration of various document contents from the perspective of an office information system rather than a typesetting system. Evolving research at Xerox PARC into document structure and integrated composition will be highlighted briefly.

### 2.5.1 Etude

The Etude project at MIT [Ilson, Etude] [Hammer, Etude] was part of a larger office automation research project. Etude (easy to use display editor) concentrated on the integration of document editing and formatting. The document formatting functionality was similar to Scribe while the internal formatting model was based on T<sub>E</sub>X's boxes-and-glue model. Etude operated on 'high level typographical objects,' such as a chapter, section, paragraph, or italic phrase. A document design database mapped these objects into formatting attributes. The system was designed to be a 'what you see is what you get' document composition system with a high standard of typographic quality, oriented towards the 'professional user.' User interface issues and minimizing training were major goals of the research effort in Etude [Good, Etude interface].

Documents in Etude are hierarchical structures. A document exists as two structures, one for the internal representation of the content and another for the representation of the formatted document broken into lines, columns and pages. The document structure has the potential for accommodating nontextual content but this has not been described in published papers. Support for mathematics and tables appears to have been deferred.

Etude supports the notion of a document style by providing formatting environments for each high level typographical object. These environments supply attribute values for formatting parameters. Relative values are permitted and the current attribute value is determined by an inheritance scheme that traverses the path from the root to the current node in the hierarchical structure.

The Etude formatter displays changes as they are entered into the document. An incremental formatting algorithm minimizes the recomputation necessary to display the changes. Extra state information is maintained in the formatted document structure to support the incremental displayer. The incremental algorithms provide the basis for developing formatters and displayers for more general document objects represented as boxes and glue.

### 2.5.2 Janus

The Janus project at IBM Research [Chamberlin, JANUS] took a slightly different approach to integrating document composition. The Janus workstation uses two different displays of the document, one the abstract manuscript file and the other the formatted document. Editing changes are made to the manuscript file and periodically the formatted document view is updated.

Janus is a declarative formatter like Scribe, rather than a procedural document compiler like `troff` or `TEX` that relies on macro packages to execute primitive formatting operations. A declarative tag on part of the document annotates the intention of an author to compose a heading or an itemized list. The document may contain images as well as text since the tag may interpret the document content as it wishes, perhaps as words of text, or as a line drawing or scanned image. Incorporating mathematical and tabular material is planned but has not yet been accomplished.

The definition of tags involves specifying the names of the tags, coding a tag action routine, and designing a page layout template. The tag markup language is a direct descendant of the 'Generalized Markup Language' of IBM's Document Composition Facility [-, DCF] [-, GML]. A document designer creates a library of tag routines that captures the formatting attributes and layout actions related to the tagged content. Tag routines are coded in a Pascal-like language. Page templates control the placement of the tagged object and are designed using a graphical design tool.

The Janus formatting algorithm is based on the boxes-and-glue model of `TEX`. Tag routines produce boxes that are collected into a galley. Out of sequence boxes, such as the text of a footnote or a floating illustration, are represented by an anchor in the galley pointing to a galley fragment. The packer algorithm places boxes from the galley into the page templates. The resulting structure is the formatted document. Janus provides for local intervention in the final positioning of boxes on a page by moving pieces of the document. Any such changes are lost when the document is reformatted.

The published papers on Janus give no specifics on how illustrations are accommodated by the tag routines. The tag routines provide a very general capability for interpreting style and rendering nontextual content. The Janus prototype was expected to provide the base for future research in formatting mathematics and tables, but no published information is available.

Janus provides insight into the customization of formatting documents by providing a formatting language for writing the tag routines to accommodate new classes of document content objects. To support a new class, one writes a tag routine and links it into the existing software. The style machinery in Janus is not centralized; each tag routine can accept its own set of attributes. This will cause difficulty with tables where the style information comes from the document containing the table, the table, and from each row and column.

### 2.5.3 Xerox Star

The Xerox Star [Smith, Star Interface] [Seybold, Xerox's Star] approaches integrated document composition from the office information perspective. Since office documents are its major focus, Star supplies less capability to describe and control all of the possible typographic features.

Nonetheless, Star integrates several classes of document objects, such as text in various fonts and sizes, simple business graphics, mathematical notation, tables, and forms. The Star user interface design is carefully managed to ensure that common actions operate across all document object classes. For instance, the act of copying part of a bar chart is the same as copying part of a mathematical formula or a part of a sentence of text.

The style mechanism in Star provides only for the specification of individual formatting attributes. There is no indirection or central registry of named collections of attributes; changing the appearance of an entire document requires changing the attributes of each instance.

Illustrations created by Star Graphics [Lipkie, Star Graphics] are mainly simple business graphics. Predefined categories of graphic images are provided: bar charts, pie charts, and simple line drawings such as organization charts. Scanned images are not supported. The centrally designed user interface extends to illustrations and its property sheet mechanism provides style attributes for graphic illustrations.

Mathematical notation is handled very well by Star. The notation is displayed in a WYSIWYG fashion using special fonts for the mathematical symbols. When symbols take their sizes from the expressions nearby, such as summation or large parentheses, these symbols grow automatically as the expressions are changed. The mathematical notations are built-in and not extensible, but cover most of the rudimentary algebraic notation.

Star also supports the interactive editing and formatting of tables and forms. Tables are defined as a matrix of rows and columns with some distinguished

entries spanning multiple columns. Table appearance is controlled through specifications in a set of property sheets for the table, the rows, columns, headings, and the rules (or lines) within the table.

Because the fundamental premise for Star assumes an office environment, there are many extensions and quality issues that were avoided. Adapting the Star to more typographically demanding environments will require addressing many of these design issues.

#### 2.5.4 Xerox PARC Research

Xerox PARC has continued to research integrated document composition systems. Prior to the development within Xerox of the Star office workstation, PARC had built the Bravo editor [Lampson, Bravo]. Bravo is an integration of both the editing and formatting of text and provides a WYSIWYG display of the document. When a document was to be printed Bravo produced a Press format file. The Press format is a device independent representation of the marks on paper. Text, line drawings, and scanned images are all treated in a resolution and device independent notation. However, Bravo could not incorporate illustrations directly and relied on the PressEdit utility to merge Press-file versions of the illustration into the document Press file.

This digression into how illustrations were incorporated in documents serves to highlight the distinction in meaning between integrated editor/formatters and integrated document composition systems. An integrated editor/formatter permits editing a text document which you see presented in the form that it will be printed; an integrated document composition system stresses the integration of various kinds of document content, typically text and illustrations, into a single integrated document.

The Tioga editing system that is part of the Cedar programming environment [Teitelman, Cedar] attempted to provide an extensible document structure to integrate document object classes. Tioga is noteworthy for its document structure and style machinery. The hierarchical node structure in Tioga evolved from NLS. Each node contains the document content and formatting properties. The editor provides interactive operations on that structure, such as selecting subtrees of nodes, displaying nodes at various depths, and applying properties to subtrees. Special node properties can be interpreted by the formatter to extend the kinds of content supported. Current artwork properties designate Press files, scanned images, line drawings, and tables.



The Tioga style machinery maps node properties into formatting attributes. Style rules are explicitly named, written in an interpreted style language, and stored in style dictionaries. Attributes for a node are inherited through the node hierarchy. Relative attributes, such as "make the indent of this node so much more than the indent of its parent," are easily accommodated. The style machinery is extensible by defining new style dictionaries and new attributes.

The implementation has not progressed beyond a robust integrated editor/formatter with some extension capabilities. Tioga served as the test bed for various experiments in document objects discussed in later chapters of this thesis.

### 2.5.5 WYSIWYG – or is it?

What is it that you get with WYSIWYG formatters? There are two answers: either a preview of the final appearance or the real appearance of the document on the output device. If what you get is the real appearance, the acronym might be renamed 'what you see is *all that* you get.' When a WYSIWYG formatter displays the real appearance, then there is no compensation for the resolution capabilities between printers and displays, sometimes a ratio as high as 20:1. In true WYSIWYG the final hardcopy output is artificially limited by the display technology. The MacWrite formatter produces almost identical printed output on the ImageWriter as the displayed output because the screen and printer fonts are the same [Seybold, MacWrite].

If what you get is a preview, then the acronym should be 'what you see is *almost* what you get.' Resolution differences result in positioning inaccuracies and limited font discrimination on the display. With limited resolution and small type sizes, there are simply too few bits to convey the distinctions between typefaces like Times Roman, Garamond, and Baskerville. Therefore WYSIWYG formatters often provide only generic typefaces that distinguish major type families and characteristics, like serif *versus* sans serif, or bold *versus* italic. The Xerox Star provides this kind of preview capability.

A WYSIWYG formatter constitutes a dilemma. On the one hand, creators of documents may tend to spend excess time and energy on the wrong aspect of document production. Instead of creating information, there is a tendency to make beautiful documents of little value. On the other hand, authors are notorious in the publishing world for making substantial *author's alterations* only after their manuscript is typeset. A typeset manuscript is more readable so authors read it more carefully and often see a different message in that context.

Significant benefits in improving the quality of documents accrue to authors working with a document in its close to final appearance, using the ability to incorporate those insights into the manuscript prior to publication.

WYSIWYG formatters are expensive. Displaying typographic fonts implies using higher resolution displays and more computational power than simple text editors. These formatters are more complex because they must maintain formatted data structures while accepting changes to the document. Operating all the formatting controls requires investing time in learning to use such a formatter, although the time can be reduced with good user interface design such as evidenced by the Xerox Star. Further study of these problems seems warranted, but is beyond the scope of this thesis. The work reported here is designed for future incorporation into WYSIWYG systems.

## 2.6 Document Content Models and Views of Documents

The representation of documents changes radically through this survey of document composition systems. Traditional graphic arts processes produce a master document only on paper or photographic film. Electronic processes produce computer files for the electronic master document with all the text, illustrations, and formatting information included. The prevalent document representation is a simple stream of text with embedded commands, as used by `troff`, Scribe, `TEX`, and Janus. This simple model is ubiquitous and may be used by preprocessing tools for these formatters. More complicated representations involving a structured document organized into a hierarchical tree or linked directed graph structure are used by NLS, Tioga, and Etude. Typically the structure contains property lists or a labelling of the content associated with parts of the structure.

Abstract document structures has been recently studied by Kimura [Kimura, thesis] [Kimura&Shaw, Abstract Documents]. This representation is a graph-like structure composed of abstract document objects. The abstract objects are mapped into concrete objects by a formatting process and these concrete objects are made visible by a viewing process. This structure has introduced ordered and unordered sets of objects, and the sharing of objects within the document structure.

Structured documents have several advantages at the expense of a more complex representation. The scope of operations can be specified in terms of substructures of the document, such as rearranging sections within a chapter, the

items in a list, or rows and columns of a table. A *class* mechanism for building extensible document object classifications can be superimposed on the document structure. Each object in the document structure can have a content class associated with it, and a specific set of procedures to perform editing and formatting operations. A distinct advantage of such a class mechanism is that all text is marked as text even if it used in the context of other objects, such as illustrations or tables. Therefore spelling checkers can check all of the text objects anywhere in the document when represented in a structured document.

More general document structures, such as integrated documents and databases, are suggested in Chapter 6. As the document structure becomes more complex, the difficulty in managing the pieces of a document increases. Almost all formatters provide a mechanism to include parts of a document within a larger document. Scribe does the best job of managing components of large documents, especially cross references. However, few schemes exist to provide interactive formatters for nonhierarchical document structures.

Another advantage of structure within the document representation is the ability to present the reader with different views of the information. Hierarchical structures permit showing only a few levels of the hierarchy (level clipping) to reveal the outline structure of the chapter and section headings. NLS also provided a view of only a few lines of each paragraph (line clipping) to compress more thoughts onto the same display space [Engelbart, NLS]. Additional views might be based on selecting content matching a pattern string or on matching properties of the reader to the nodes in the document structure [van Leunen, One Document]. Cargill's notion of different views of software based on configuration properties [Cargill, Views] demonstrates how a single comprehensive structure, possibly containing redundant information, may be accessed to produce several different configurations of a document.

# 3

## Graphical Style

---

### 3.1 Producing High Quality Illustrations

This chapter addresses the problems of producing high quality illustrations in documents. In particular, the issues of ensuring stylistic consistency among a group of related illustrations and of reusing illustrations in different media or for different purposes are presented. The concept of *graphical style* is introduced to deal with both of these problems. The illustrations considered here are two-dimensional images formed with lines, curves, areas, scanned raster images, and text. Three-dimensional images must currently be reduced to two-dimensional representations for displaying on screens or printing in documents. The concept of graphical style however extends gracefully to structured image rendering systems of any dimension.

The research into graphical style for illustrations was conducted at Xerox PARC in 1982. That work resulted in a paper coauthored with Maureen Stone and presented at the 1983 ACM SIGGRAPH conference [Beach&Stone, Graphical Style]. The paper reports on a prototype implementation, TiogaArtwork, in the Cedar programming environment [Teitelman, Cedar] using the Tioga document structure and the CedarGraphics imaging software [Warnock&Wyatt, CedarGraphics]. This chapter provides additional background to the problem and summarizes the main features of the graphical style prototype.

Few of the document composition systems outlined in Chapter 2 can integrate illustrations within the electronic manuscript file. Those that can impose severe restrictions on the achievable quality: `pic` and `ideal` produce

line drawings with only a single line weight, and the Xerox Star does not output to typeset-quality devices. Illustrations for many typeset documents are prepared by skilled draftsmen and then manually pasted onto the final typeset pages. In the past, computer-generated illustrations have been easy to identify by their poor quality, especially in author-prepared manuscripts that feature diagrams prepared on pen-plotters with irregular character shapes.

This is not a new problem brought on by computer technology, but one which has always existed. All too often the illustrations in a book, especially in a technical book, do not receive the same production treatment as the text. Some publishers require the author to prepare the illustrations while others provide minimal resources for creating illustration artwork.

“When the author’s contract stipulates that he is to supply illustration copy, he may choose to draw it himself or get it drawn by somebody else whose main qualification for the task is that he will make no charge for it, or next to none. The resulting material may be clear enough to explain its meaning but incapable of adequate reproduction or too irregular in drawing to appear in a well-produced book.” [Williamson, Book Design, p 258]

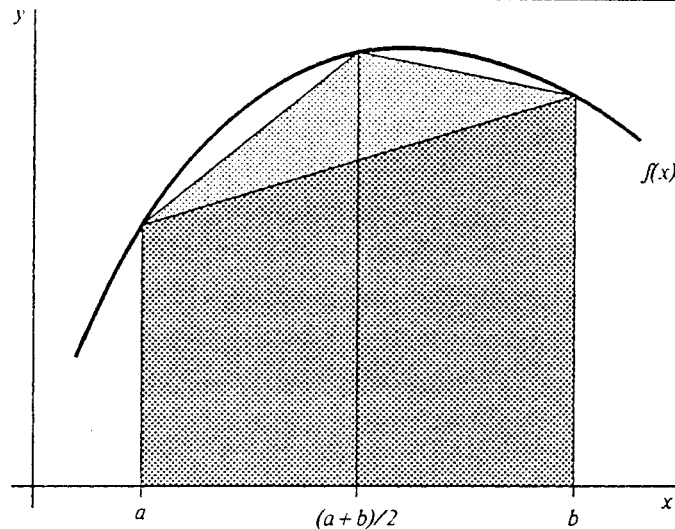
The incentive for this research was to develop tools for use by an author who chooses to draw illustrations himself yet wishes to achieve a graphic arts standard of quality.

### 3.1.1 Text Book Illustrations

Two college text books [Dyck, PASCAL] [George&Liu, Sparse Matrices] formatted and typeset by the author of this thesis are representative of the difficulties one encounters in producing publishable quality illustrations. Both books are heavily illustrated: the first book contains over 150 line drawings and the second about 80. Both books contain hundreds of computer program fragments and many mathematical equations, typographic requirements that are often difficult and expensive when preparing text books without electronic composition tools. The computer program fragments could be treated as text and simply typeset with an appropriate font to resemble the fixed-width characters on a computer line printer. The mathematical equations were formatted with a system similar to eqn. However there were no support tools for preparing the illustrations which were ultimately drawn by hand and pasted onto the final pages.

The numerous illustrations in the two text book projects could be grouped into a few categories: mathematical graphs of curves complete with axes, tick marks, and labels; syntax charts for a programming language; data structure

diagrams consisting of shaded boxes for variables and curved arrows for pointers; sparse matrix layouts depicting an arrangement of nonzero elements; and schematic diagrams to illustrate examples or exercises at the end of the chapters. Ensuring that the draftsman prepared all of the illustrations from a particular category in a consistent manner proved difficult. Several factors varied: choice of line weight, reduction percentage for illustrations drawn larger than finished size, positioning and style of tick marks on graphs, treatment of points along a curve, treatment of arrows and arrowheads, and the typography of text labels on graphs. Figure 3-1 is an example of one of the mathematical graphs from the *Computing* text book.



*Trapezoidal Rule for  $n=1$  and  $n=2$ .*

Figure 3-1. TRAPEZOIDAL RULE FIGURE from *Computing* [Dyck, *Computing*] redrawn with the Griffin illustrator using a style faithful to the hand-drawn original. This illustration will be used in several examples in this chapter. (Used with permission from Reston Publishing Co.)

To control this variation, guidelines were drawn up to establish the desired choices. An axis line was always to be drawn with a thin line; the data curve would be drawn with a heavier line; points along the curve would be dots of a certain radius; tick marks would be a specified length; arrows would bend a preferred way and have a certain kind of arrowhead. Text captions and labels proved to be the most troublesome to handle. The book designers had taken great advantage of the flexible electronic document composition system to use several fonts in a disciplined way. To ensure that the illustration labels conformed to the typography of the book, the labels had to be typeset separately and supplied to the draftsman, who either cut and pasted the labels onto the

artwork, or else rubbed on customized transfer lettering developed from the typeset labels.

The goal of those efforts was to ensure a consistent appearance among all the illustrations in each category. Books on graphic design urge this discipline and consistency:

“Every item in the book gains in appeal to the reader’s eye from its relationship with all the other items. Something of a family resemblance, an appearance of being a set of pictures rather than a collection from disparate sets, may confer this advantage on the illustrations of any edition.” [Williamson, *Book Design*, p 256]

The first problem is to *ensure consistency among a set of illustrations*. In developing the illustration style guidelines, several iterations between designer and draftsman were needed to specify the correct rules completely. It became obvious that this iterative design process was essentially the same process as specifying the book design guidelines between a graphic designer and the programmer of a document formatting package. The consistency problem for illustrations would be solved if one could capture and share style guidelines for rendering computer-generated illustrations in precisely the same way that style guidelines are developed for text preparation.

The second problem is to *extend the lifetime of illustrations* beyond a simple use. Both of the text books were used in college courses. Lecturers in those courses wanted to present overhead transparencies of the illustrations. Text book publishers often supply instructors’ manuals free of charge with transparency masters for creating overhead slides. All too frequently to save time and money, the preparation of these transparency masters is done near the end of the book production cycle (or afterward) from inexpensive typewritten illustrations. It would be preferable to have the same illustrations from the text book available as transparency masters for overhead slides.

In the published forms of the two text books, the graphs and diagrams used fine lines and shaded textures appropriate for the typeset material surrounding them. Those same fine lines and shades do not reproduce well onto transparencies. For overhead or 35 mm slides projected to a large audience, the lines must be much darker, the text much bolder, and the shaded textures much coarser or (better still) displayed in color. The problem of reusing illustrations across different media would be greatly reduced if one could automatically render the illustrations with different graphical attributes suitable for different media.

If it were easier to reuse illustrations, then perhaps people would take more time and care to create really good ones.

### 3.2 Previous Work

Most earlier computer-based illustration systems have been primarily concerned with functionality. The thrust has been in developing algorithms for synthetic graphics and in implementing software to render illustrations. Many of these systems permit graphical attributes to be assigned to elements of the images [Crow, Image Environment], but they have no mechanism to collect attributes for easy change.

The PICTURE language [Beatty, Picture] and Picc [White, Picc], a later implementation for the C language, define picture description languages for publication quality illustrations. The style of the illustration is described by graphical parameters supplied to the rendering operations, such as line width and dash pattern required by the line drawing routine. These parameters provide explicit control over some aspects of graphical style. The TELE-A-GRAF business graphics package by ISSCO [–, TELE-A-GRAF] provides templates for generating idiomatic graphics with a convenient style of presentation built into the template. Two other interesting illustrators, JUNO [Nelson, Juno] and GOB [Zabula-Salalles, GOB], use constraints to create pictures that preserve certain relationships after changes in portions of the picture, but there is no style capability to make a set of illustrations consistent.

Interactive illustrators, such as those developed at Xerox PARC, also suffer from a lack of style facilities. Bitmap illustrators like MARKUP [Newman, Markup] draw lines of different widths or shade areas with different textures to create a black and white bitmapped illustration. However, the illustration has a fixed resolution and one cannot easily change the line widths or shading textures after the illustration is created. An object illustrator like DRAW [Baudelaire, Draw] makes it easier to manipulate graphical objects, but does not provide attributes for all the rendering options nor a means of grouping graphical attributes for objects that look similar. The Griffin illustrator [Baudelaire&Stone, Griffin] does have an explicit notion of style attributes and provides menus for selecting line weight, filled and/or outlined areas, colors, text fonts, sizes, and orientations. Still, there is no grouping of style attributes nor any naming or indirection to allow sharing of attributes among several objects with the same style.



The `troff` preprocessors for illustrations, `pic` [Kernighan, `pic`] and `ideal` [van Wyk, `ideal`], can produce simple line drawings. There are few attributes for the graphical objects and no style mechanism. The line drawing capabilities are device specific, provide only a single default line weight, and are implemented on some typesetters by overlaying a prodigious number of dots to form connected curves.

The Xerox Star provides a comprehensive integration of graphics within office documents [Lipkie, Star Graphics]. The editing and formatting environment of the Xerox Star permits similar user interaction techniques to be used across both graphical and textual material. In particular, formatting attributes for both textual and graphical objects are assigned through the same property sheet mechanism. However, there is no grouping or indirection of these properties.

Consider an example of the frustration that occurs when there are graphical attributes but no style mechanism which occurred at Xerox PARC. A Griffin illustration of three roses was selected for use in a trade show demonstration of a new graphics printer. The original illustration had red flower petals with green leaves and stems. Unfortunately, the new printer could produce only three shades of grey and no color. The printing software substituted the same grey pattern for both red and green colors. The result was a rather flat picture. To change the colors of the red petals, green stems, and green leaves to three distinct greys required tediously applying style attribute changes to each petal, leaf and stem of the three roses by selecting each one individually. It would have been much easier if the petals, leaves, and stems each referred to a named set of graphical attributes that could be changed once to affect all references.

Grouping and sharing graphical attributes into styles is not a new idea. An early proposal by Thomas in 1976 for specifying display parameters in graphics programming languages [Thomas, Graphics Parameters] contains the essence of the graphical style idea. The Graphical Kernel System [ANSI, GKS] also provides a mechanism for grouping graphical attributes into 'bundles' that are assigned to graphical objects to be rendered by a display workstation. However, these ideas have never been integrated into document composition systems or document style mechanisms.

### 3.3 The TiogaArtwork Prototype

The TiogaArtwork prototype illustration system was an experimental implementation of graphical style. An extended document structure that

incorporates illustrations and an extended style machinery for graphical style attributes were embedded into the existing Tioga document composition system in the Cedar programming environment [Teitelman, Cedar]. The Cedar graphics package [Warnock&Wyatt, CedarGraphics] provided the necessary graphics rendering algorithms, including drawing straight lines and curves with different thicknesses, shading areas with various colors or textures, typesetting text with graphic arts fonts in various sizes, and rendering continuous tone images from either scanned or synthetically computed sources.

The TiogaArtwork illustration system is integrated with the Tioga document formatter in two ways: the document structure was extended to include both text and illustration objects, and the style machinery was extended to incorporate graphical formatting attributes. The resulting document structure for text and illustration objects provides a recursive nesting of text within illustrations and illustrations within other illustrations. Such an integrated document structure provides the basis for integrating illustrations into the editor, since many of the user interaction techniques can be made similar for text and graphics, in the fashion of the Xerox Star user interface. The style machinery extensions define additional graphical style attributes needed by the illustration rendering algorithms.

These extensions to the document structure and style machinery provide the basis for separately specifying the *form* (or rendering) of an illustration from the *content* (or geometry) of an illustration.

### 3.3.1 Tioga Document Model

The document model in Tioga is a tree structured hierarchy of nodes, much like NLS [Engelbart, NLS]. Textual documents are typically organized as paragraphs within sections within chapters. Each document node has textual content and an associated property list. The properties of a node affect the formatting algorithms by supplying parameters or hints.

---

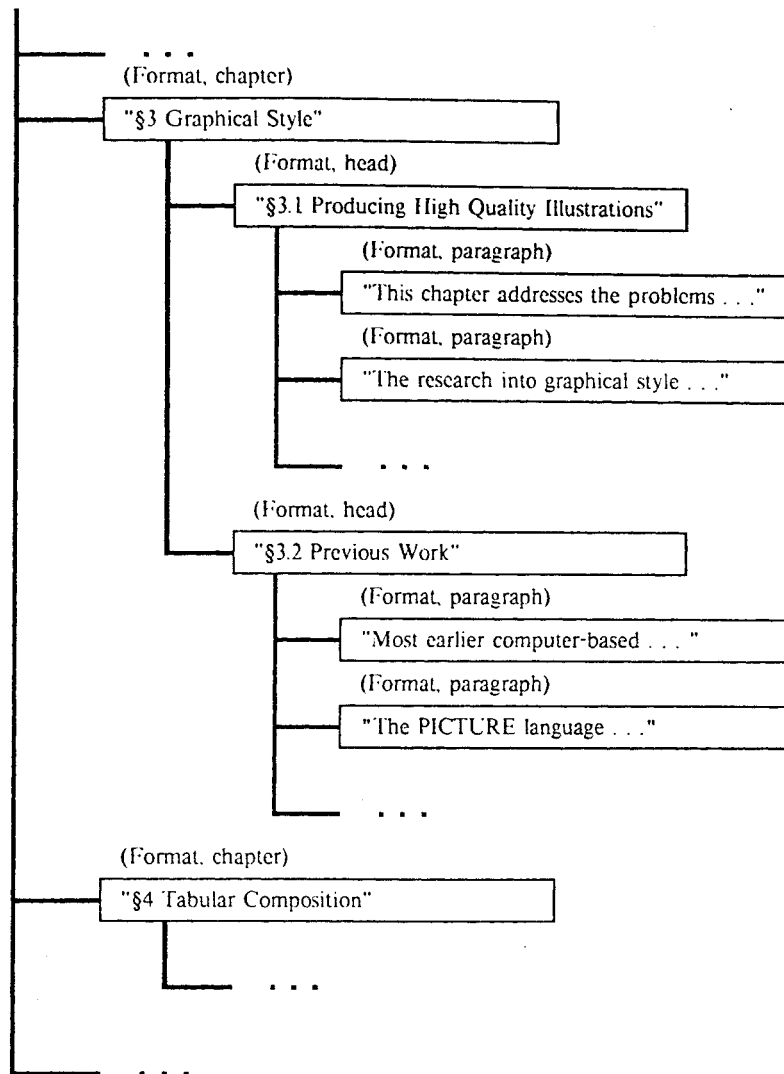
 (Style, WaterlooThesis)


Figure 3-2. THE TIAGA DOCUMENT STRUCTURE is a hierarchical structure of text nodes. Each node is shown as a text phrase enclosed in a box. The hierarchical structure is shown by lines connecting boxes: lines down indicate sibling relationships (several section headings within a chapter); lines to the right indicate children (chapters contain section headings that in turn contain paragraphs). Each node has a property list shown in parentheses on top of the node box. The root of the structure has a property, (Style, WaterlooThesis), that defines the style dictionary appropriate for formatting PhD dissertations. Formatting style rule properties on each text node identify a group of formatting attributes.

---

Two standard properties for Tioga nodes are `Style` and `Format`. (Property names and values are distinguished by a special typeface in this chapter). The `Style` property identifies a dictionary that binds style rule names to formatting attributes. There may be any number of style dictionaries, corresponding to kinds of formatted documents such as `Cedar` for program source code files, or `BlueAndWhite` for Xerox PARC Technical Reports (which happen to have blue and white covers). The `Format` property identifies a style rule in the current style dictionary. The style rule name for a particular node is usually chosen to relate to some semantic notion in the document, such as `paragraph`, `head`, or `item`.

Formatting attributes in the Tioga style machinery are defined by formatting algorithms which register an attribute name and a default value. Style rules are sets of interpreted instructions that assign values to the formatting attributes. For instance, the `paragraph` style rule sets the type size for normal text, the `head` rule increases the type size and sets the font bold, and the `item` rule increases the left indent.

The hierarchical path from the root to the node forms a search path for locating attribute-value bindings similar to programming language scoping. A `Style` property establishes the identified dictionary as a current scope for all nodes in the subtree spanned by that node. Attributes are assigned their values by walking the search path and executing each style rule in turn. A formatting style rule is found by searching for the style rule in the set of nested scopes. An extensive caching mechanism in the Tioga style machinery makes this traversal efficient. The existing Tioga style mechanism deals with about 50 text formatting attributes.

These format properties are analogous to declarative tags in Janus or Etude, and the formatting attributes they describe are similar to properties in the Xerox Star. However, unlike NLS and Tioga, those other document composition systems lack an explicit structured document model.

### 3.3.2 Artwork Class Nodes

To extend the Tioga document model to incorporate illustrations, a new node property, `ArtworkClass`, was defined to distinguish the illustration node content from plain text. The `ArtworkClass` property is interpreted by the document formatter which treats the node content as the specification of an illustration.

During development of the prototype, the Tioga editor was not modified and thus did not recognize the new `ArtworkClass` property. The content of these artwork nodes was a textual representation of illustrations. Therefore, the Tioga editor displayed them as text and could not present the images in a WYSIWYG fashion. Future implementation outlined in Chapter 6 will involve modifying the editor to provide a general mechanism for WYSIWYG editing classes of nontextual nodes, including illustrations.

We next define the representation within an `ArtworkClass` illustration. Illustrations have a natural hierarchy for positioning subpictures relative to other pictures as recognized in the earliest computer graphics systems [Sutherland, Sketchpad]. This hierarchy is mapped directly onto the Tioga document model, one subtree for each subpicture. The positioning in the hierarchy is accomplished by providing a stack of transformations and activating a new positioning transformation at each branch in the tree. These are the standard graphical transformations of translation, scaling, and rotation. The illustration content is formed from the set of graphical primitives: lines, curves, areas, scanned raster images, and text. Each of these is defined as a distinct class, described below.

The class mechanism permits recursive inclusion of content. Thus a text document may contain an illustration, and that illustration may in turn contain a text label. Furthermore, illustrations may contain any future class of object, such as mathematical equations or tables.

The artwork class of nodes is formatted using an object-oriented design. Each artwork node is represented by an object with two associated procedures, one for layout and one for rendering. The layout procedure is given the document subtree rooted at the current document node and returns the bounding box dimensions of the formatted artwork object. The dimensions of the box may be specified as glue [Knuth, *The T<sub>E</sub>Xbook*], with appropriate stretch and shrink, so that the object can be included in the existing page layout algorithms just as a normal box, but with a special rendering procedure. The rendering procedure is given the subtree and the dimensions determined by the layout procedure. It produces an instance of the object at that size on the formatted output stream, normally a printable document file.

The TiogaArtwork prototype implementation defines several artwork classes. A registry mechanism permits extensions to be added by supplying the necessary layout and rendering procedures for each additional class. Objects of the first artwork class, `ArtworkNode`, serve as the roots of subpicture trees in the document structure and normally contain the transformations for positioning the

(Style, BeachThesisArtwork)

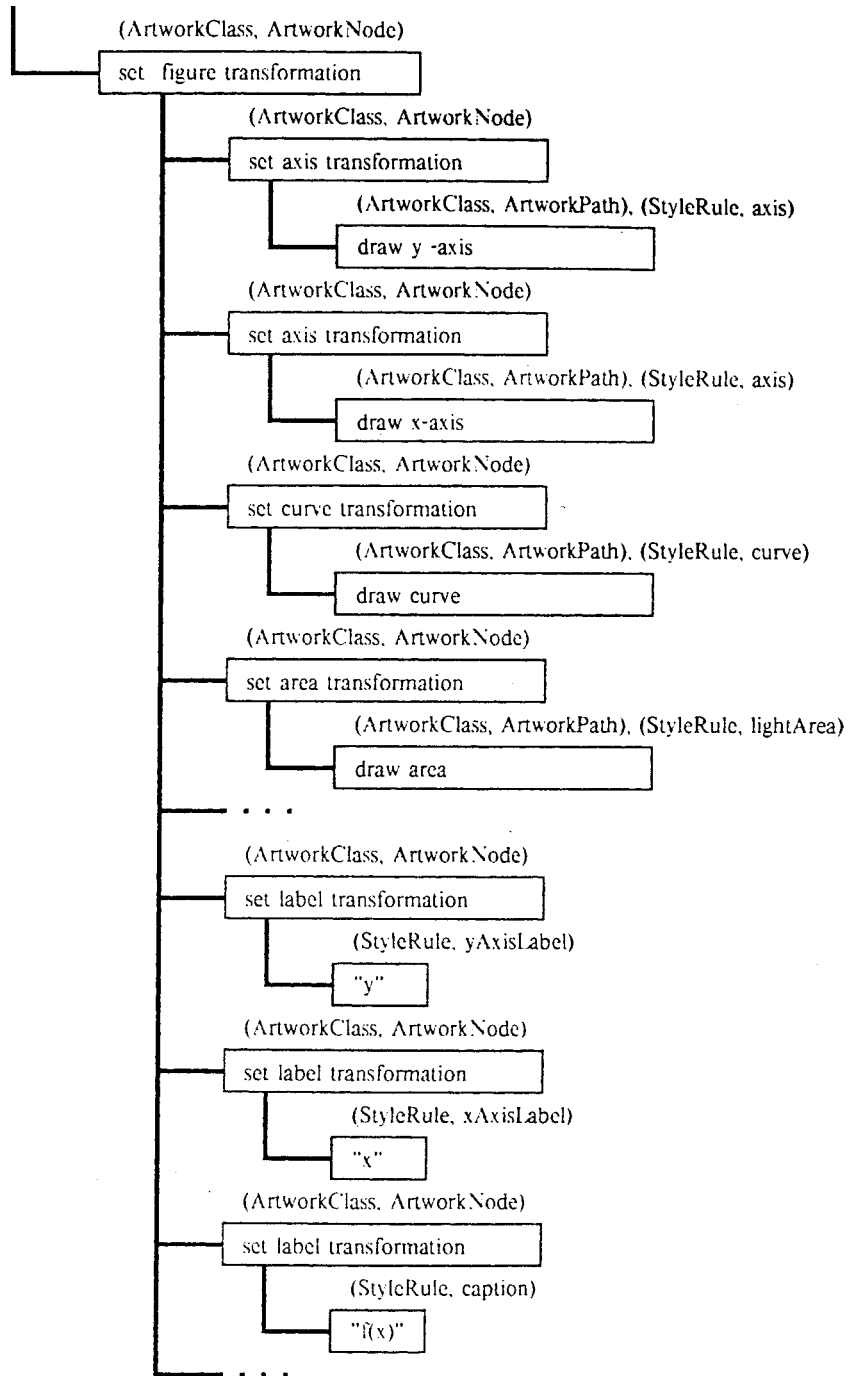


Figure 3-3. HIERARCHICAL ILLUSTRATION STRUCTURE of Figure 3-1. The ArtworkClass property identifies a graphical object and a description is shown as the content. Note the inclusion of text nodes, without an ArtworkClass property, for the caption labels.

subpicture. An `ArtworkPath` class node contains the geometric definition of a path for a line or curve object, described in detail below. An `ArtworkImage` class node defines a continuous-tone scanned image. In the prototype implementation of the image class the node content is the filename of the scanned image, although a future extension would be to include the scanned image data directly in the node. An `ArtworkFileName` class node contains the name of another TiogaArtwork illustration file and thus serves as an inclusion mechanism for large or shared illustrations. The prototype provided no additional naming capability for sharing subpictures, although such a facility could be added.

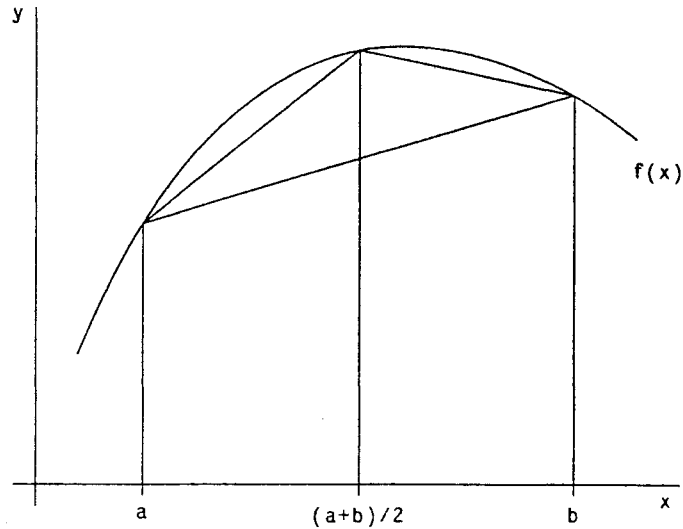
### 3.3.3 Geometric Representation of Illustrations

The internal geometric representation of illustrations in the TiogaArtwork prototype is based on a text description in an interpretive graphics language in Cedar [-, JaM]. The language is stack-oriented with postfix operators for arithmetic, logical, and graphical operations. Geometrical objects are defined by a *path* that determines the trajectory followed by a line or a curve, or the boundary of an area filled with color or texture. All of the rendering parameters for the geometric objects in an illustration are defined by the style rules described in the next section. Figure 3-4 contains the plain geometry of the trapezoidal rule illustration. The textual representation of this illustration is shown later in Figure 3-5.

Transformations in an illustration can be composed of translation, scaling and rotation elements appropriate for standard graphics packages. The hierarchical structure for the illustration is traversed using a standard tree-walk. At each branch in the tree, the current transformation is stacked and the new transformation concatenated. These operators are used to specify the positioning transformations:

```
x y .translate - translate the origin to <x,y>  
sx sy .scale - scale by the factors sx in x and sy in y  
r .rotate - rotate clockwise by r degrees
```

The paths that define stroke trajectories or areas are composed of straight lines and Bézier parametric cubic curves. The graphics package provides the notion of a current point that is set at the beginning of the path and updated as line segments and curves are added to the path. As a trajectory, the path specifies the centerline of strokes. As a filled area, the path specifies the boundary of the area. Note that no rendering specifications are necessary in the



Trapezoidal Rule for  $n=1$  and  $n=2$ .

Figure 3-4. SKETCH OF THE ILLUSTRATION for Figure 3-1 represents the basic geometry of the picture. The same TiogaArtwork representation was used for this illustration as for Figure 3-1, however all of the rendering attributes have been reduced to drawing only thin lines and using a typewriter-like typeface.

geometrical specification, because they are available through the style machinery. These operators are used to define geometrical paths for each synthetic graphical object:

$x\ y$  .moveto – establish the current path position  $\langle cx, cy \rangle$  as  $\langle x, y \rangle$  with respect to the current transformation

$x\ y$  .lineto – extend the path with a straight line from the current path position  $\langle cx, cy \rangle$  to  $\langle x, y \rangle$ , and update the current path position  $\langle cx, cy \rangle$  to be  $\langle x, y \rangle$

$x1\ y1\ x2\ y2\ x3\ y3$  .curveto – extend the path with a curve which has the four Bézier control points  $\langle cx, cy \rangle$ ,  $\langle x1, y1 \rangle$ ,  $\langle x2, y2 \rangle$ , and  $\langle x3, y3 \rangle$ , and update the current path position  $\langle cx, cy \rangle$  to be  $\langle x3, y3 \rangle$



---

```

% TiogaArtwork figure for Trapezoid Rule
% Cluster 1
0 0 .translate 1 1 .scale 0 .rotate
  % y-axis
  11 31 .translate 1 1 .scale 0 .rotate
    1 1 .moveto 1 185 .lineto
  % x-axis
  3 39 .translate 1 1 .scale 0 .rotate
    1 1 .moveto 249 1 .lineto
  % curve
  27 87 .translate 1 1 .scale 0 .rotate
    1 1 .moveto
    8 17 15 33 25 49 .curveto
    43 78 71 106 105 113 .curveto
    131 118 161 110 185 97 .curveto
    194 92 201 87 209 81 .curveto
  % area from a to (a+b)/2
  51 39 .translate 1 1 .scale 0 .rotate
    1 1 .moveto 1 97 .lineto
    81 161 .lineto 81 1 .lineto
    1 1 .lineto
  % area from (a+b)/2 to
  ...
  % y-axis label
  8 216 .translate 1 1 .scale 0 .rotate
    y
  % x-axis label
  247 36 .translate 1 1 .scale 0 .rotate
    x
  ...

```

---

Figure 3-5. GEOMETRIC REPRESENTATION of Figure 3-1 in a textual form consists of transformations and path definitions. The Trapezoidal Rule illustration was first drawn with the Griffin illustrator and then automatically converted into a TiogaArtwork representation. The node structure corresponds to Griffin clusters and the node properties correspond to groups of Griffin style attributes. Here, the indentation indicates the node structure. The style properties are not shown. Comments, which begin with percent signs, were added for exposition purposes only by manually editing the text.

---

### 3.3.4 Graphical Style Attributes

Extending the style machinery for the TiogaArtwork prototype required defining additional style attributes to provide parameters for the various graphical rendering algorithms. Some of these attributes specify straightforward parameters, such as line thickness or area color. Other attributes specify how the geometry of the illustration should be treated, for instance, whether the path defines an outline or an area or both. Secondary attributes are necessary to describe how outlines are to be drawn. For instance, a pen metaphor, similar to the one in METAFONT [Knuth, METAFONT], is used to draw lines that provides different shapes of pens specified by various parameters. The following attributes are provided in the TiogaArtwork prototype for simple line drawings. Some values are keywords and others are numeric. Later in this chapter, Figure 3-7 contains several examples of graphical style rules.

`pathType` – the choice of path treatment as an area or outline or both:  
           filled, outlined, filled+outlined

`lineWeight` – the line thickness (a measurement)

`penType` – the choice of pen shape: round, square,  
           rectangular, elliptical, italic

`penHeight` – the pen height as a proportion [0..1] of `lineWeight`

`penWidth` – the pen width as a proportion [0..1] of `lineWeight`

`penAngle` – the rotation of the pen, in degrees clockwise from  
           horizontal

`areaColor` – the color of filled areas as hue, saturation, brightness  
           values in the range [0..1]

`outlineColor` – the color of outlines as hue, saturation, brightness  
           values in the range [0..1]

Additional attributes define how textual captions and labels should be handled within illustrations. Several attributes come directly from the existing Tioga style attributes, while others were added to distinguish among the variety of caption alignments:

`family` – the name of a type family, such as "Helvetica" or "Times  
           Roman"

`size` – the type size (a measurement)

- `face` – the choice of type style: `regular`, `italic`, `bold`, and `bold+italic`
- `captionFormat` – the choice of text justification format: `flushLeft`, `flushRight`, `centered`, or `justified`
- `captionAlign` – the choice of the text alignment point: `flushTop`, `centered`, `baseline`, or `flushBottom`
- `lineLength` – the length of caption lines (from Tioga)
- `leftIndent` – the left indent for captions (from Tioga)
- `rightIndent` – the right indent for captions (from Tioga)
- `leading` – the spacing between lines of text (from Tioga)
- `textRotation` – the rotation of the text line, in degrees clockwise from horizontal
- `textColor` – the color of caption text: hue, saturation, brightness values in the range [0..1]

### 3.3.5 TiogaArtwork Rendering Algorithms

Recall that artwork class objects have two procedures, one for layout and one for rendering the illustration. The layout procedure returns the bounding box for the illustration, and the rendering procedure creates an image either on a display screen or in a printable file.

In fact, both these object procedures depend on the rendering algorithm. The bounding box information for layout is collected by rendering the illustration through a special imaging device that computes the bounding box of all the graphical objects it sees. The document formatting algorithm accepts the illustration as a box with those dimensions and positions the illustration box within a page. With the position determined, the formatter invokes the object rendering procedure to create the viewable illustration.

The rendering technique is to walk the illustration subtree in prefix tree-order, and for each child of the subtree root representing a subpicture, stack the current transformation, render the subpicture and pop the transformation stack. The subpicture node class determines the layout and rendering procedures to use for an object of that class.

*Algorithm A (Render Artwork)*

- Λ1 [Traverse the illustration subtree] Given the root of the illustration subtree, walk the tree. For each child node of the root:
  - Λ1.1 [Stack Current Transformation] Request the graphics package to remember the current transformation on its stack.
  - Λ1.2 [Invoke Node Render Procedure] Select the object rendering procedure depending on the class of object found. Common classes are listed here for convenience.
    - Λ1.2.1 [ArtworkNode] Concatenate the transformation and invoke Algorithm A on this node as the root of a subpicture.
    - Λ1.2.2 [ArtworkPath] Render the path according to the graphical style attributes using the geometry defined by this node as needed.
    - Λ1.2.3 [ArtworkImage] Render the raster image stored in the file named in the node contents.
    - Λ1.2.4 [Text] Format the text caption using the text class layout procedure and graphical style attributes to position the caption.
  - Λ1.3 [Pop Transformation] Pop the transformation stack in the graphics package to return to the transformation of the parent node.

The path rendering algorithm uses the graphical style attributes to determine how to use the geometrical path information. Given the separate specification of geometry from style in a *TiogaArtwork* illustration, one can make multiple uses of the geometry to create special effects. Multiple use has already been discussed for areas that are both outlined and filled where the path serves once as the boundary of the filled area and a second time as the centerline of the outline. Other special effects, such as shadows, arrow designs, and border patterns, were considered for the prototype. Only shadows were implemented in *TiogaArtwork*, although techniques for rendering arrow and border designs along path geometries were discussed in the Graphical Style paper [Beach&Stone, Graphical Style].

Two types of shadows were implemented to simulate apparent depth: a drop shadow, where the object is drawn with a slanted shadow, and an offset shadow,

where the object is drawn repositioned slightly from the original and in a different color. A similar scheme was used for highlighting text in an interactive paint program previously developed by the author of this thesis [Beach, Paint].

The shadow style attributes defined were the following:

- `shadowType` – the choice of shadow effect: `drop` or `offset`
- `shadowAngle` – the angle of the shadow from the object, in degrees clockwise from the horizontal
- `shadowDirection` – the direction of the shadow from the object: `upLeft`, `upRight`, `downLeft`, `downRight`
- `shadowPathType` – the choice of offset shadow treatment: `filled`, `outlined`, `filled+outlined`
- `shadowOffsetAmount` – the distance that the offset shadow is placed at `shadowAngle`
- `shadowWeight` – the thickness of the drop shadow or the outline of the offset shadow
- `shadowAreaColor` – the color of the drop shadow or offset shadow area
- `shadowOutlineColor` – the color of the offset shadow outline

### 3.4 Results

To experiment with the graphical style prototype, line drawing illustrations were created with the Griffin illustrator and converted into the TiogaArtwork document structure. The conversion program understands the Griffin file format and extracts the geometric definition of objects from the files. Griffin illustrations can be built with clusters of graphical objects and the clusters are preserved as a subpicture tree in the hierarchical TiogaArtwork document structure. The coordinate origin for each cluster is set to the lower left corner of its bounding box and an appropriate transformation is inserted into the root of the subpicture tree. Griffin style attributes for each object are collected into TiogaArtwork style rules and a style dictionary is built automatically for each illustration. Generic names for the style rules and style dictionary are synthesized by the conversion program. These names can be edited by hand to reflect more meaningful names, and the styles for several illustrations collected into a single style dictionary. This process simulates the operation of an

interactive WYSIWYG-style user interface for an illustrator program (one that was not built as part of the prototype experiment). The trapezoidal rule figures in this chapter, Figure 3-1, Figure 3-4, and Figure 3-6 were all created this way.

Existing scanned images, new images scanned using services on the network, or raster images computed by image synthesis algorithms may also be included as illustrations. TiogaArtwork nodes referencing image files or illustration files were inserted into test documents 'by hand' for the prototype using the Tioga editor.

The class mechanism for artwork illustrations proved to be a very successful extension strategy for Tioga documents. Several additional `ArtworkClass` properties have been implemented through this mechanism, including tables discussed in Chapter 5.

The TiogaArtwork scheme was successful in creating pictures more complex than previous interactive illustrators at PARC. In particular, the concept of transformations and named subpicture elements stored in a hierarchical fashion was not available in the Griffin illustrator. The inclusion of simple text is supported by Griffin, but the general formatting capabilities of Tioga are only accessible through the TiogaArtwork scheme. The combination of synthetic line drawings with images and formatted text within a document was not available, except through a 'paste-up' program for combining printer format files.

The TiogaArtwork prototype was successful in separating the graphical attributes from the illustration rendering algorithms. The graphical style concept thus provided the enforcement mechanism to ensure that a set of illustrations in a document have a consistent appearance. Reusing the same illustration for different media was also made possible by changing the graphical attributes in alternate versions of the style rules. Figure 3-6 contains the same trapezoidal rule illustration as Figure 3-1, but with a style suitable for a 35 mm color slide presentation. Figure 3-7 contains the two sets of style rules for the typeset illustration, Figure 3-1, and the 35 mm slide illustration, Figure 3-6.

Unfortunately, the style concept is not sufficient to capture all the notions of changing illustrations across media. Unless the illustration and its style attributes are carefully designed, such as carefully choosing the text alignment and anchor points, unfortunate results may occur when changing the style without changing the content. When an illustration is prepared as a projected slide there is less room for detail. Suppressing detail is a change supported by some graphic systems, such as in Crow's scene assembler [Crow, Image Environment], which the prototype graphical style system does not accommodate. Text size is a style

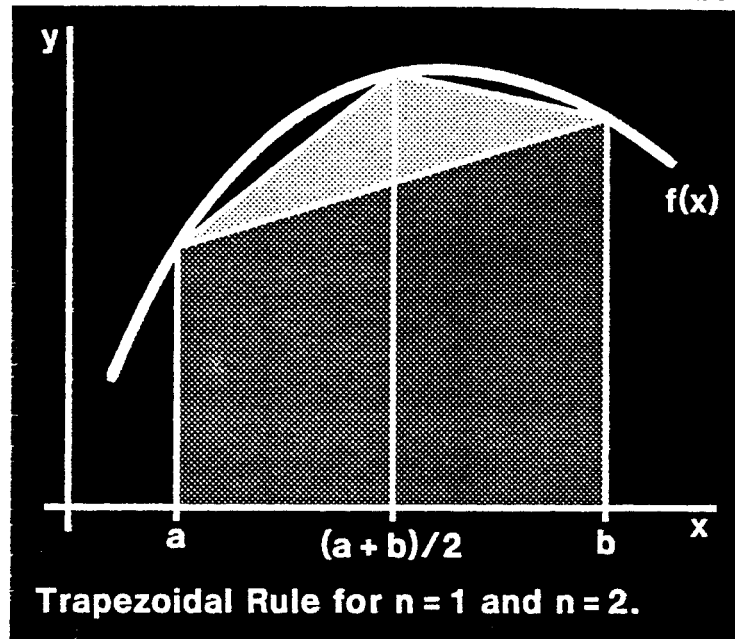


Figure 3-6. TRAPEZOIDAL RULE SLIDE uses the same picture file as Figure 3-1, but with a style appropriate for a projected 35 mm color slide. The image has the 'preferred' format with light detail on a dark background, thicker lines in white, a larger, bolder and simpler typeface. (Used with permission from Reston Publishing Co.)

parameter, but transformation scaling of the graphical objects is not. For instance, the graphical box surrounding a text phrase would not change size when the text was made larger. Constraints would be an asset in changing the graphical objects with respect to surrounding material. Incorporating the constraint aspects of Nelson's JUNO illustrator [Nelson, Juno] into the illustration artwork rendering system might alleviate these problems. This is discussed in Chapter 6.

Manipulating styles in the TiogaArtwork prototype stresses the tools available in Tioga. More interactive tools, such as property sheets from the Xerox Star, would be a boon to selecting attributes and naming format rules. New style tools should list the set of style dictionaries available, list the formatting style rules in a selected dictionary, and list the values of formatting attributes defined in a selected style rule. Style rules could then be defined to be the 'same as but different' from other style rules by naming one style rule and setting specific attribute values to be different. Layout parameters might be more easily specified through an interactive design tool for styles.

The goal of the graphical style research was to provide a new tool for the graphic artist to be more effective when mixing illustrations within electronic

---

% TrapezoidBook.Style	% TrapezoidSlide.Style
BeginStyle	BeginStyle
(BasicGraphics) AttachStyle	(BasicGraphics) AttachStyle
(BasicText) AttachStyle	(BasicText) AttachStyle
(axis) "x,y axes" black outlineColor outlined pathType 1 pt lineWeight StyleRule	(axis) "x,y axes" white outlineColor outlined pathType 2 pt lineWeight StyleRule
(darkArea) "dark areas" grey areaColor filled pathType StyleRule	(darkArea) "dark areas" orange areaColor filled pathType StyleRule
(lightArea) "light areas" lightGrey areaColor filled pathType StyleRule	(lightArea) "light areas" lightYellow areaColor filled pathType StyleRule
(curve) "function line" black outlineColor outlined pathType 2 pt lineWeight StyleRule	(curve) "function line" white outlineColor outlined pathType 4 pt lineWeight StyleRule
(caption) "text caption" "TimesRoman" family 8 bp size italic face flushLeft captionFormat flushTop captionAlign 0 leftIndent black textColor StyleRule	(caption) "text caption" "Helvetica" family 12 bp size bold face flushLeft captionFormat flushTop captionAlign 0 leftIndent white textColor StyleRule
(xAxisLabel) "text label" caption center captionFormat StyleRule	(xAxisLabel) "text label" caption center captionFormat white textColor
(yAxisLabel) "text label" caption flushRight captionFormat StyleRule	(yAxisLabel) "text label" caption flushRight captionFormat white textColor
EndStyle	EndStyle

---

Figure 3-7. GRAPHICAL STYLE SHEETS for the two Trapezoidal Rule illustrations in Figure 3-1 and Figure 3-6 demonstrate the style language and the graphical style attributes. The style on the left produces a typeset book quality illustration and the style on the right produces a colored 35 mm slide form. Note that the styles differ in the choice of line weights, color selections, and typography parameters, and that the caption rules depend on a common definition with different alignment. The style rules are named for the obvious parts of a mathematical graph.

---



documents. Just as careful document design permits a manuscript to be published in several different forms, the careful design of illustrations and their styles leads to the ability to reuse illustrations for several purposes and to control the production of high-quality illustrations more efficiently.

# 4 Tabular Composition

---

## 4.1 What is a table?

The previous chapter concentrated on illustrations and provided a style mechanism for the graphical information within documents. We now turn our attention to the problem of laying out the arrangement of information in two dimensions. Table formatting presents a concentrated form of this problem. It has a strong two-dimensional nature because tables are composed of entries arranged into rows and columns.

This chapter defines a general notion of a 'table' and surveys early work in typesetting tables. It then describes the typography of tables and tabular composition. The final section reviews the capabilities for typesetting tables in existing electronic document composition systems. A good summary of tabular formatting in the graphic arts is contained in Phillip's article "Tabular Composition" published in *The Seybold Report* [Phillips, Tabular Composition].

A *table* is an orderly arrangement of information. Tables are defined to be 'rectangular arrays exhibiting one or more characteristics of designated entities or categories' [-, Dictionary]. Tables may be less structured than this, simply serving to present a list of entries. However in most cases, tables have some structure that is relevant to the presentation of information. We will take a fairly general view of tables, encompassing a broad range of layout possibilities.

Within this view, the layout of mathematical notation might be considered a small instance of table formatting and page layout might be considered a large

instance of table formatting. These comparisons will be elaborated in Chapter 6, where we will argue that the table formatting framework presented in Chapter 5 can be extended to deal with both of these other layout problems.

The succinctness of a table aids in revealing and understanding complex relationships within the information.

“Tables offer a useful means of presenting large amounts of detailed information in small space. A simple table can give information that would require several paragraphs to present textually and can do so with greater clarity. Tabular presentation is often not simply the best but the only way that large quantities of individual, similar facts can be arranged. Whenever [the] bulk of information to be conveyed threatens to bog down a textual presentation, an author should give serious consideration to use of a table.” [—, *The Chicago Manual of Style*, 1982, p 321]

Designing table typography is a hard problem. There are many formatting details to get right and there is only a small amount of space with which to work. The two-dimensional nature of tables requires alignment in both directions at the same time. It is very important to maintain control over placement because the organization of information in tables is part of the message. Juxtaposition and other spatial relationships within tables have an important impact on the way in which tables convey information.

“The principles of table making involve matters of taste, convention, typography, aesthetics, and honesty, in addition to the principles of quantification.” [Davis, *Tabular Presentation*, p 497]

There seem to be great opportunities for applying electronic techniques to typesetting tables of information:

“Tabular setting has proved both the easiest and the most difficult form of composition to bring under computer control. Because tabular setting is mainly for numeric data, it might seem strange that there should be any difficulty in providing computer-generated [typeset] tables.” [Phillips, *Handbook*, p 189]

However, tables in technical documents contain a wider variety of information than the traditional mathematical tables of roots, logarithms, and trigonometric functions.

“While many tables of physical and scientific data are being compiled by computer, there is still a requirement to include these data in technical publications because they are considered of interest to the reader who may

not have access to the generating algorithms even if he is a computer user. The publication of such data in printed form may also be considered necessary to establish the status of the author! It would appear that the need for tabular composition in general bookwork will continue for some time." [Phillips, *Tabular Composition*]

The sources and purposes of tables in documents span a broad range of information. Some examples include computed data from mathematical algorithms, statistical data from scientific experiments, financial data and *spreadsheets*, taxonomies of observed data, extracts from databases of information, or just about anything else an author might wish to convey to a reader.

## 4.2 Early Table Formatting Systems

As mentioned in Chapter 2, several early composition systems could produce typeset tables. Computers were heavily involved in numeric computations at that time. Publishing tables of numeric results by traditional methods required the error-prone transcription of line printer or punched card data by keyboard operators of typesetting devices. Because photomechanical typesetting devices used electronic input data that were compatible with computer systems (mainly punched paper tape and occasionally magnetic tape), it was natural to conceive of a computer program that would convert the numeric data directly to the formatting commands suitable for driving the typesetting devices. These commands could then be passed directly between the computer system and the phototypesetter.

Several reviewers [Barnett, *Computer Typesetting*] [Stevens, NBS99] [Phillips, *Handbook*] have reported that the earliest book of computer typeset tables was the monograph produced at the National Bureau of Standards by Corliss and Bozman in 1962 [Corliss&Bozman, NBS53]. Those tables of numeric calculations were formatted by a special program running on an IBM 7090 computer and were output onto tape for a Linofilm phototypesetter. The tables included column heads in bold type centered over numeric data aligned on decimal points. Reportedly, this monograph contained only a single tabular format throughout [Stevens, NBS99, p 6].

Another pioneering effort in typesetting tables was TABPRINT [Barnett, *Computer Typesetting*] developed by Barnett at MIT in the early 1960's. TABPRINT ran on an IBM 7090 computer and prepared tapes for a Photon 560

phototypesetter. The tabular data was input in a fixed format typical of numeric computations in that era. Typographic specifications for each table preceded the set of data records and provided rudimentary style capabilities. Several typographic refinements to the program were proposed as future developments, including such features as folding long column heads over narrow data fields, introducing blank lines every 5 or 10 lines, and grouping digits of long numeric values for readability.

These programs for formatting tables of numeric data were relatively simple. "The significance of this early work in tabular composition is that all the typographic parameters were defined by program." [Phillips, Handbook, p 195] However, tables of numeric data constitute only one aspect of table formatting problems:

"But there are really two very different categories of tabular composition: One comprises a book of similar tables in which the values shown can be calculated by program algorithms from the minimum of data input, and the other consists of the tables appearing in technical texts. In the first case the style is similar for many consecutive pages, but in the second case each table, and there are sometimes several tables on the same page, has different column widths, different numbers of columns, and also ranges the entries differently, both vertically and horizontally; in addition, each table may have different complex box headings." [Phillips, Handbook, p 189]

To format the more general table designs required in technical publications, we need effective interactive design tools that can handle a wide range of typographic requirements. Interactive tools seem preferable because many table designs are unique. The variety of table designs limits the amortization period for the time invested in programming a table formatter with sufficient specifications to accomplish each arrangement. Phillips expected interactive table composition programs to be necessary because of the typographic complexity of tables:

"These complications will tend to keep interactive terminals employed for page make-up and with soft-copy proofs on page view terminals." [Phillips, Tabular Composition, pg. 23-11]

The next section investigates these complications and the typographic requirements for formatting aesthetic tables.

### 4.3 Typographic Requirements for Tables

#### 4.3.1 Tables are Two-Dimensional

Tables have a two-dimensional structure because of the organization of the table into rows and columns. These row and column structures intersect to identify the characteristics of the table entry at the intersection. The layout of a table must simultaneously align table entries horizontally in a row and vertically in a column. The table width is determined by accumulating the widths of each column. In turn, column widths are determined by the widths of the entries in the column. Similarly, the table and row depth are determined by the depths of the entries in each row. The arrangement of table entries can be expressed separately from the actual widths of rows and columns. This separation of the topology (ordering) of entries from the geometry (positioning) of entries will be exploited in Chapter 5.

The two-dimensional nature of tables differentiates table formatting from simpler text formatting. Tables deal with areas and graphical relationships, both of which have two degrees of freedom. Lines and paragraphs of text have only one degree of freedom (where to break the line), although even then a complex algorithm may be necessary to produce aesthetic line breaks [Knuth, Line Breaking].

The conventional two-dimensional structure of tables is illustrated in Figure 4-1. The rows and columns intersect to form the table entries in the *panel* which is the main body of the table. The area with row identifications at the left of the panel is called the *stub*. The column headings in the area along the top of the panel are called the *box head* because when a table is fully outlined the column heads are completely boxed. Some headings group several columns together and are referred to as *spanning heads* or *spanning subheads*, depending on the depth at which they occur in the box head. Spanning row headings for several rows are also possible.

As an example, the box head in the table of Figure 4-1 has completely determined the width of each column because the headings themselves are wider than the information in the columns. In other tables, the table entries may be wider than the headings above them, and they would then determine the column widths.

Stub Head	Spanning Head				Col. Head
	Col. Head	Spanning Subhead		Col. Head	
		Col. Head	Col. Head		
Row Head	xxx	xxx	xxx	xxx	xxx
Row Head	xxx	xxx	xxx	xxx	xxx
Total line	xxx	xxx	xxx	xxx	xxx

Figure 4-1. THE TWO-DIMENSIONAL STRUCTURE OF A TABLE includes the arrangement of its entries into rows and columns. Here the parts of a table have been shaded for easy identification. The light grey area is the *box head* that contains all of the column headings. The dark grey area is the *stub* that contains all of the row identifications. The remaining white area is the *panel* containing the actual table entries.

Various graphic embellishments to the basic row and column structures help convey the table information. Dividing lines called *rules* help separate dissimilar parts of the table. The box head and stub in Figure 4-1 are completely outlined; all possible horizontal and vertical rules are present in the headings. Some table designers prefer only horizontal rules (see below for more discussion of table rules).

The word 'rules' will appear frequently in this and the next chapter, in relation to the typographic lines (rulings) drawn in a table to separate rows or columns. This use of the word is traditional in the graphic arts. However, it may be confused with the notion of 'style rules' from the previous chapter. Throughout this thesis, the word 'rule' by itself refers to a typographic line and terms 'style rule' and 'formatting rule' refer to a way of doing things.

The content of table entries may vary considerably. Certainly textual and numeric information are commonly organized into tables. Other types of information often included in tables are pictures, illustrations, mathematical equations, and even other tables.

In most table designs, the table entries are fully contained within the row and column intersection. More general table designs permit the content of one table entry to flow into another. Connected entries would be necessary when folding a long table entry of text into two column entries, or when flowing a caption around several illustration entries. This capability is necessary to extend table formatting to full page layout requirements.

### 4.3.2 Typographic Treatment

This section discusses the wide range of typographic details required to format tables. Careful text placement is an obvious requirement stemming directly from the two-dimensional structure of tables. Alignment choices to guide the placement are also needed. Formatting attributes can be applied to different parts of the table structure. The treatment of whitespace, typographic rules, and rows of dots between table entries are devices for guiding the eye along rows or columns. Footnotes on table entries must be referenced and positioned appropriately. Finally, readability concerns in table formatting are important.

#### *Fine Resolution Placement*

Compared to the line lengths in normal text, table entries are formatted to a relatively short line length within each column. These short line lengths force more hyphenation and line breaks in text entries. Placing several narrow columns side-by-side requires inserting some whitespace between each column to improve readability. Centering table entries or balancing space between entries also requires fine control of their position. These small distances must be chosen carefully since the human visual system perceives patterns and groupings, whether intentional or not, and bad choices may dramatically affect the way the table is interpreted by a reader.

Typesetting devices often provide resolutions in very small units, a common one being 1/10 of a printer's point (about 1/720 of an inch). Formatting a table with the coarse positioning typical of a fixed-pitch printer or typewriter is a much easier task. The results are normally not very aesthetic, since the fixed-size units are quite large, but they eliminate many choices and decisions:

“Tabular material is always difficult to typeset – much more so than to compose on the typewriter. This is true even though figures have a ‘monospaced’ value. Letters do not, and therefore it is more difficult to align material or even to determine what will fit in a given space . . . The monospaced typewriter – where you can actually visualize what you are setting – is certainly the simplest way for the novice to proceed. And it will not be an easy task for the typesetter to imitate what the typist has done.” [Seybold, Fundamentals]

#### *Alignment within Tables*

The alignment choices within tables correspond to the two-dimensional nature of table layouts. Horizontal row and vertical column alignments predominate. However, other alignments for spanned headings, equal width



rows or columns, and balancing the extra whitespace between columns are common.

Column entries are *vertically aligned* with each other in various ways, as seen in Figure 4-2. (Note that one must adjust an entry horizontally in order to align it vertically with another above or below it; such distinctions are made carefully in the remainder of the thesis.) The three most frequent choices for vertical alignment are flush to the left (generally for textual material), flush to the right (generally for numeric material), or centered within the column (generally for headings and textual material).

FlushLeft	Center	FlushRight	Decimal.Align
xxxxxx	xxxxxx	xxxxxx	000000
xxxxxxxxxxx	xxxxxxxxxxx	xxxxxxxxxxx	00.000000
xxxx	xxxx	xxxx	.000
xxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxx	0000.0

Figure 4-2. VERTICAL ALIGNMENT WITHIN A COLUMN of table entries is commonly flush left, flush right, centered, or decimal aligned.

Numeric data with a varying number of decimal digits require another type of vertical alignment where the data items align on the decimal point. Numeric entries without decimal points must have one inferred, usually after the last decimal digit. The alignment on decimal points can be generalized to alignment on any character. For example, mathematical equations are often aligned on their equality signs. More complex alignment possibilities arise when multiple alignment points are needed, such as aligning the terms of polynomials in a system of equations where each of the additive and subtractive operations require alignment (although the unary minus sign does not):

$$\begin{array}{rcl}
 10x_1 - 7x_2 & = & 7, \\
 -3x_1 & + & 6x_3 = 4, \\
 5x_1 - x_2 + 5x_3 & = & 6.
 \end{array}$$

Just as for columns, row entries are *horizontally aligned* with each other in various ways, as shown in Figure 4-3, again with three frequent choices, flush to the top, flush to the bottom, or centered. (Again, note that an item is adjusted vertically to accomplish horizontal alignment.)

Flush Top	xxx xxxxxxxxx xxxx	xxxxxxx xxxxxxx	$xx^x + yy_y + zz$
Center	xxx xxxxxxxxx xxxx	xxxxxxx xxxxxxx	$xx^x + yy_y + zz$
Flush Bottom	xxx xxxxxxxxx xxxx	xxxxxxx xxxxxxx	$xx^x + yy_y + zz$

Figure 4-3. HORIZONTAL ALIGNMENT WITHIN A ROW of table entries is commonly flush top, centered, and flush bottom as indicated by the stub labels on each row. The entries in the second and third columns have multiple lines of text. The last column contains entries with superscripts and subscripts that affect the height and depth of the text entry. Alignment without regard to baselines produces unaesthetic results, especially when centering an even and an odd number of lines, and when aligning entries with different heights and depths. A fine point: note that the capitalization of the stub labels affects their position when aligned.

Row entries possess an additional characteristic similar to decimal-point alignment: the baseline on which successive characters are aligned. The rightmost column of the table in Figure 4-3 contains entries with baselines different from the other three columns. Without a horizontal alignment choice for baseline alignment, table entries with different baselines will not be arranged in a visually pleasing manner. This problem is addressed in Chapter 5.

Spanned headings are aligned within a set of columns, or set of rows if the heading spans several rows. The set of columns spanned by the heading determines the aggregate dimensions of the spanned heading. Should the heading exceed this size, perhaps because it is longer than the narrow columns it spans, then the heading may be folded to make it shorter, or the columns spaced out to accommodate the long heading. Spanned row headings have similar needs.

Equal widths of columns (or equal heights of rows) may be called for. In some cases the precise size will be specified by the designer and applied to the table. In other cases, the size can be determined automatically by the largest entry in the set of rows or columns.

### *Formatting Styles*

Tables are often formatted with a different (but related) set of attributes to those used for normal text. Frequently tables are typeset in the same typeface but in a smaller point size, both to attract less attention to the table and to include more information. These changes in formatting attributes promote the use of a separate formatting environment or set of style rules for tables.

Further specification of formatting attributes is necessary when rows or columns are to be distinguished. For instance, a row of totals may be the most important aspect of the table and therefore should be set in a bolder type face, or one column of information may be exceptional and thus be distinguished in an italic type face. Finally, individual table entries may be distinguished with special formatting attributes such as highlights.

### *Whitespace Treatment*

The treatment of whitespace between table entries is more complicated than between paragraphs of text because there are more relationships for each table entry. The space between two columns of text is called the *gutter* in normal formatting, while the space between table entries is generally referred to as the *bearoff* or *bearoff distance*. The separation of rows or columns with whitespace helps to establish the apparent grouping of data. The introduction of rules into a table permits the physical separation to be reduced or eliminated since the grouping is provided by the rule.

Some strategies for compacting large tables to fit a page (discussed later) involve shrinking the bearoff space. The bearoff may provide a place for a footnote reference or *gloss* marker to intrude between table entries without expanding the column width. These markers do not participate in the alignment of table entries and therefore need not be separated with the same bearoff distance.

Excess whitespace due to a large spanned heading requires apportioning the space among bearoffs for the spanned rows or columns.

### *Rules and Decorations*

The use of dividing rules within tables to separate rows or columns is a traditional practice. Rules run along the row or column boundaries in either the horizontal or vertical direction. In large tables with narrow columns, vertical rules are often indispensable in maintaining order among the vast quantity of

data. The preference for horizontal rules is a recent phenomenon due in part to faddish design preference and in part to harsh economic reality. Consider the experience of the University of Chicago Press by comparing the statements from the 1969 and 1982 editions of *The Chicago Manual of Style*:

“Ruled tables, for example, are usual in the publications of this press, in part because Monotype composition has always been readily available. For a publisher who is restricted to Linotype, open tables or tables with horizontal rules alone may be the only practical way tabular matter can be arranged.” [—, *A Manual of Style*, 1969, p 273]

“In line with a nearly universal trend among scholarly and commercial publishers, the University of Chicago Press has given up vertical rules as a standard feature of tables in the books and journals that it publishes. The handwork necessitated by including vertical rules is costly no matter what mode of composition is used, and in the Press’s view the expense can no longer be justified by the additional refinement it brings.” [—, *The Chicago Manual of Style*, 1982, p 326]

The difficulty with inserting vertical rules stems from the mechanical properties of photocomposition devices. With manual makeup of pages from metal type, inserting rules involved laying down a thin metal strip. High-speed phototypesetting devices that have only a narrow aperture across the page are strongly biased towards the horizontal, both for typesetting text and for drawing typographic rules. This same bias towards the horizontal is reflected in the composition software that supports these devices. Newer typesetting devices with more accurate positioning of laser beams can print in both orientations with equal ease and eliminate this restriction.

There are several distinguished rules that frequently occur in tables: the *head rule* above the box head, the *cutoff rule* below the box head, the *spanner rule* below a spanning head, the *foot rule* below the table, and the *total rule* above the total row. These rules may be of different thicknesses, with the outermost head and foot rules generally drawn thicker than rules inside the table.

Rules come in a variety of shapes, sizes, and patterns. Different thicknesses or *weights* of rules provide appropriate emphasis. A common design is to use medium-weight rules for the head and foot rules above and below the table, and fine *hairline* rules for the cutoff rules between the column headings and the table entries [Williamson, *Book Design*, p 159]. Double rules or combinations of thick

and thin rules are sometimes used to provide emphasis and closure to a table. The intersection of these patterned rules is a complicated affair.

Braces that group table entries are sometimes required within tables. The brace is placed in the space between two rows or columns, sometimes requiring extra space to accommodate its curly shape. Braces are frequently added by hand from transfer lettering sheets because they are not supported by table formatters and their positions are awkward to specify and align properly.

*Ornaments*, such as flowers or other interesting designs, are inserted at the corners or along the outer border of a table. They are old fashioned and used mainly as a decoration for the purpose of catching the reader's attention.

*Background tints* were used in Figure 4-1 to highlight the different parts of the table. Traditionally, tints would be added by hand at the page makeup or camera stage since they involved halftone screens. Phototypesetters and laser printers can produce screens automatically by shading the area of the table before the content is typeset.

### *Leaders*

Various graphic techniques, such as *dot leaders*, help the reader capture the content and meaning of the table.

*Leaders* are the dot patterns that guide your eye from an item at one side of a table to the related item at the other side of a table. Headings in tables of contents are often connected with dot leaders to the page numbers on the right. Typically, leaders are formed from dots although dashes or rules are sometimes used. Dot leaders are positioned congruently so that successive rows of leaders all have the dots in the same horizontal position. The harmony of the aligned dots enhances their purpose of guiding the reader without distraction. Leaders cross through column gutters and possibly vertical rules, although rules are ill-advised when leaders are used.

### *Footnotes within Tables*

Footnotes within tables pose an interesting layout problem. As in page layout, footnotes for table entries are collected and placed at the bottom of the table within the page area allocated to the table. This means that for the table formatter to accommodate footnotes, it must be at least as powerful as the page formatter. Most table formatters only handle footnotes placed manually within the table.

By convention, footnote references are separately marked or numbered for each table. Typically, footnote references within tables use letters or symbols rather than superscript numbers to avoid confusion with numeric exponents in the data. Should footnote references be numbered, they usually are sequenced independently from any text footnotes.

### *Readability Issues*

Tables of numeric information have been published for many years and there are classic methods for making tables more readable [Knott, Napier commemorative]. For example, long columns of numbers are separated with extra whitespace or with thin rules every 5 or 10 entries to provide 'chunks' that help the human visual system scan the long columns. Background tints behind rows of a table are another technique to improve readability in long tables. Grouping digits in threes with commas or extra whitespace provides the same chunking for long decimal expansions of logarithms or trigonometric functions.

### **4.3.3 Large Tables are Awkward**

Tables tend to be awkward to handle in page composition. They must be treated separately from the running text because they contain separate information. However, the tables may be too wide for the page width or too long for the remaining space on the page, or even too long for the page height. Following are some of the problems and solutions for dealing with large tables.

### *Common Strategies for Large Tables*

Tables are commonly formatted in a smaller type size to reduce the impact of the table on the reader. This choice also helps fit more information in a table. Reducing the point size to 70% or 80% of the text size reduces the character height and width proportionately. Common sizes for text are 10-point type on 12-point leading. Tables often use 8-point type on 9-point leading or even 7-point on 8-point. Compressed type faces have the same height but reduced width that permits more text in the same horizontal space. For example, Helvetica Light Condensed is a narrow font commonly used in tables.

The bearoff distances between table entries can be reduced to eliminate whitespace and thereby reduce the width and height of a large table.

Transposing rows into columns and vice versa [Williamson, *Book Design*, p 159] may make a large table fit the page. Wide tables with many columns are transposed into longer tables with fewer columns, and long tables with few

columns are transposed into wider tables with many columns. A table and its transpose are shown in Figure 4-4. Note that the stub heads and spanning heads have been transposed in a nontrivial matrix transposition that preserves the column heading relationships. One must be careful about transposing statistical tables that might imply an incorrect cause and effect relationship [Zeisel, Figures, p 41].

Stub Head	Spanning Head				
	Col. Head	Col. Head	Col. Head	Col. Head	Col. Head
Row Head	xxx	xxx	xxx	xxx	xxx
Row Head	xxx	xxx	xxx	xxx	xxx
Row Head	xxx	xxx	xxx	xxx	xxx

Spanning Head	Stub Head		
	Row Head	Row Head	Row Head
Col. Head	xxx	xxx	xxx
Col. Head	xxx	xxx	xxx
Col. Head	xxx	xxx	xxx
Col. Head	xxx	xxx	xxx
Col. Head	xxx	xxx	xxx

Figure 4-4. TRANSPOSING A TABLE may help make a table fit on the page. The top table is wide with more columns than rows. The bottom table is the transpose of the top table and is narrower with fewer columns than rows.

### *Long Tables*

Some tables can be made shorter by folding a long column into multiple columns. For instance, one long list of names in a single column would become two or more lists of names. This folding trades off shorter table length with increased table width.

Long tables that exceed the page height must be broken into smaller tables. Breaking a table is similar to breaking lines of text at page boundaries, and similar algorithms [Plass, Optimal Pagination] can be applied. However, broken tables must introduce continuation headings in the second and subsequent parts of the table. The continuation headings may be very complicated functions of the table entries:

“It would be asking rather a lot of a page make-up program to insert carried forward and brought-forward totals automatically at a table break, and indeed these were often omitted when tables were made-up by the hand compositor.”

[Phillips, *Tabular Composition*, p 23-11]

The continuation headings can be supplied in the table input as variants of the regular headings. When a table is broken then these variations can be used. Brought-forward totals could be supplied automatically when the table structure and content is recognized within the formatting program, for example, in financial spreadsheets. This is an instance of a particular table entry (a total) that might compute itself on behalf of the table formatter (for the current total of all formatted entries). An extensible table content structure, such as that described in Chapter 5, provides a general mechanism for incorporating self-totalling table entries and other continuation headings.

### *Wide Tables*

A table that is wider than it is long may be made to fit the page by rotating the table and printing it *broadside*. A broadside table has the long dimension of the table along the long dimension of the page, that is, rotated 90° so the rows read up the page and the columns read from left to right. Right-hand pages are preferred for such tables since a turned book will present the broadside table closer to the reader [Williamson, *Book Design*, p 271]. Broadside tables (or illustrations) impact page composition, because these pages are typically designed with page numbers in a different position and without running heads (otherwise the page numbers would appear in a different orientation to the broadside table and detract from the readability of the facing page).

Instead of rotating the entire table to make a wide table fit the page, it may be sufficient to rotate the text of column headings to read vertically. Especially when the column headings are much wider than the column entries, turning the text so that it reads upwards with successive heading lines to the right reduces the column width. If column headings in a broadside table are turned, they should instead have the descenders to the left, otherwise the text would appear upside down on the page [Williamson, *Book Design*, p 159].

Wide tables may be formatted as a two-page spread across two facing pages. A two-page *upright* table would appear with the box head spread across the *binding gutter*. A two-page broadside table is possible with the rows split across the gutter. Continuation headings may not be needed in a two-page broadside table, but would be if the table continued onto subsequent pages.



Extremely wide tables may be printed on a foldout plate. This requires special paper to be folded and inserted into the book at the binding stage. The extra manual handling makes this alternative very expensive and rarely used.

Otherwise, wide tables are broken into smaller table parts with continuation stub headings. Any spanning headings in the box head will have to be continued across the break. Some reference columns, such as sequence numbers, may be repeated to assist in finding information in the continued table parts.

## 4.4 Previous Approaches to Table Formatting

### 4.4.1 The Typewriter Tab Stop Model for Tables

The use of a fixed width character model for table formatting is a key simplification available with typewriters or line printers. The fixed width of each character on these devices permits a coarse grid with complete specification of the character positions. Spreadsheet programs take advantage of this to provide regularly spaced grids and simple typographic features. There are fewer possibilities for positioning and aligning characters when using a fixed grid, making the formatting problem much simpler. The typewriter tab stop model is often provided in document composition systems as a rudimentary table formatting capability.

Tab stops are based on a physical escapement mechanism in mechanical typewriters. The carriage in old typewriters is spring-loaded and advanced to the next character position whenever a key is pressed. The tabulator key permits the carriage to fly to the right past several character positions until stopped by a mechanical 'finger.' These fingers are the tab stops. Any number of them can be requested along the carriage. The measurement between stops is always in units of character positions. Furthermore, the stops indicate only the left margin of a tab column. Aligning numeric information or centering headings requires spacing the carriage manually. Teletype devices also have tab stops, but they standardized on 8 characters between stops to ensure that the sending and receiving devices would place characters in the same positions. Early computer terminals also have 8-character tabs, while later ones have settable tab stop positions.

Document formatters extended the typewriter tab stop model to align numeric and centered information. Defining a tab stop requires specifying a formatting attribute for the position of the stop and for the alignment choice.

Different formatters choose to interpret the tab stop differently as determining a position (Runoff) or a column (Scribe). The entries in Figure 4-5 are aligned at tab stops according to the different interpretations made by the Runoff class of formatters and by Scribe.

---

Runoff/troff	left	right	center
	↑	↑	↑
Scribe	left	right	center
	↑	↑	↑

---

Figure 4-5. TAB STOPS are interpreted differently by various document composition systems. The first row treats a tab stop as an alignment position for text; the text is aligned at the tab stop. The second row treats a pair of tab stops (or a tab stop and the page margin) as defining a column within which text is aligned.

Defining a column to be the space between two tab stops creates an inconsistent notion of a column. Two short pieces of text can be positioned in the same column if the first is left-aligned and the second right-aligned; two longer pieces of text will be positioned in different columns. The tab stop defining a column does not imply a boundary, only an alignment point. A long line of text is not folded when the text extends beyond the next tab stop. Thus editing the text entry may result in aligning it in a different column than before.

Tab stops provide only a very limited table formatting functionality with few typographic features. They are not satisfactory for most tables, yet they are the only table formatting capabilities offered by several document composition systems. The tab stop model breaks down completely when table entries must be folded from one line to the next. The next section discusses the first real table formatter available with an electronic document composition system.

#### 4.4.2 `tbl` Preprocessor

The `tbl` table formatter [Lesk, `tbl`] for `troff` is a preprocessor that accepts a table definition and generates formatter commands to render the table. The table definition is in two parts: the table arrangement part and the table content part. These parts may be intermingled to keep the arrangement definition close to the affected content. The last row definition is reused whenever more row contents are encountered than defined in the arrangement part.

The table arrangements may include spanned headings across arbitrary columns or rows. However, the specification of spanned column headings is

asymmetric to spanned row headings. The spanned column heading is specified in the table arrangement part while the spanned row heading is specified in the table content part or the heading, as discussed in Chapter 2.

Folded table entries are possible with the column width stated explicitly. Whole paragraphs or complex formatted objects may be included as a table entry. Should the table contain objects formatted by another `troff` preprocessor, the order of processing must be carefully chosen to avoid interaction between the preprocessors.

Formatting attributes may be specified to apply to all entries in a given column. No similar capability is provided for rows, although `troff` commands may be inserted before and after rows to control some of the formatting attributes.

Rules may be specified within the table in a stylized fashion. Thin single and double rules may be specified; `tbl` computes the intersections between single and double rules that are only a single thickness and color. There is a shorthand specification to box all table entries. Rules are specified in an asymmetric fashion where vertical rules are included with the table topology and horizontal rules are included with the table content.

The position of table entries is computed by `troff`. `tbl` assigns the width and height of table entries to `troff` registers and uses `troff` commands to compute the position of table entries. Thus, tables are limited in complexity by the number of available registers, which is in turn limited by the two-character naming restriction.

Long tables with repeated rows can be formatted with `tbl`. Distinct continuation headings can be supplied explicitly or the original box head can be repeated for each table fragment on successive pages.

The `tbl` preprocessor has the generality to accommodate most table arrangements. The specification of spanned row and column headings permits general layouts. The asymmetric treatment of rows and columns often impacts the ease of table specification. The content of tables is limited by `troff` resource restrictions and by the interaction with other `troff` preprocessors such as `eqn` and `pic`. Recursive content is not possible. For example, a `troff` document cannot contain both a table of mathematical equations (`tbl` includes `eqn`) and a display equation that includes a table of equation fragments (`eqn` includes `tbl`), since the `troff` preprocessors must be executed in a sequential pipeline order.

### 4.4.3 T<sub>E</sub>X

There is no table formatting preprocessor for T<sub>E</sub>X. Equivalent functionality is provided through extensive macros [Knuth, The T<sub>E</sub>Xbook, Chapter 22]. The T<sub>E</sub>X `halign` (horizontal alignment) primitive defines a template for the table layout that specifies a separate formatting environment for each column. Successive rows of the table then match entries in the preamble. Complications arise when introducing horizontal and vertical rules. Sophisticated knowledge of T<sub>E</sub>X is required to master them [Knuth, The T<sub>E</sub>Xbook, Chapter 22].

The L<sup>a</sup>T<sub>E</sub>X macro package [Lamport, L<sup>a</sup>T<sub>E</sub>X] provides a specification language similar to `tbl` for defining tables. The table topology is defined as successive rows of column entries with vertical rule codes. The L<sup>a</sup>T<sub>E</sub>X scheme provides a more robust implementation than `tbl` since it is based on the T<sub>E</sub>X box and glue abstraction, whereas `tbl` simulates this abstraction with `troff` macros. The difference is revealed in the success with integrating mathematical notation in tables. L<sup>a</sup>T<sub>E</sub>X provides a more reliable and predictable table formatter compared to `tbl` and `eqn` which may fail in unexpected ways when interactions between the two preprocessors occur.

However, the L<sup>a</sup>T<sub>E</sub>X table formatting macros do not ensure that aesthetic layout is easily accomplished. The L<sup>a</sup>T<sub>E</sub>X manual cautions authors of complex tables that some final hand tuning of the space around boxes will be required to achieve the best results [Lamport, L<sup>a</sup>T<sub>E</sub>X, p 105].

### 4.4.4 TABLE

None of these previous approaches provide interactive table design capabilities. They are all batch-oriented formatters. The TABLE editor [Biggerstaff, TABLE] is a prototype interactive graphics editor developed for editing complex structures. The prototype editor provides an interactive front end to the `tbl` formatter as part of an experiment in object-oriented programming. The objects implemented were tables and text. Operations on objects would be determined by the nature of the objects, such as deleting a row or column from the table. Editors were to provide WYSIWYG feedback so that the screen image would be identical to the printed form. Several software engineering concerns about object-oriented programming were tested in this experiment, such as the ease of building and modifying an editor, and the responsiveness of editing interactions. The results favor the first two criteria but raised some concerns about the latter.

The TABLE prototype is a WYSIWYG editor for table structures. The table layout is presented graphically. The layout of table objects is accomplished with

the `tbl` preprocessor. This provides TABLE with sufficient layout generality but with the associated performance penalty of using batch programs. The paper recommends developing a custom post-processor for a production system [Biggerstaff, TABLE, p 343].

The user interacts with TABLE objects through a selection mechanism. The granularity of table selections are the entire table, a table element, or selections within the object contained in the table element. Positioning commands allow the user to traverse the table structure along rows or columns and from the table to within the table element objects.

The TABLE prototype succeeded in providing a graphical interface to complex table structures. Table designs could be generated more quickly and more accurately with TABLE than by coding `tbl` commands directly. However, TABLE inherited from `tbl` the resource restrictions, the lack of object structure characteristic of `troff` preprocessors, and the sluggish performance of a large batch formatter. TABLE lacks operations suited to the logical structure of tables through limitations in its internal table data structure and its selection mechanism. There is no style provision in TABLE, possibly because `tbl` did not provide one to be inherited.

# 5

# A New Framework for Tabular Composition

---

## 5.1 The Interactive Table Formatting Problem

This chapter presents a new framework for formatting tables that is suitable for use in interactive editors and formatters. The framework extends the object-oriented approach of the document structure model in Chapter 3 by providing for the arrangement of objects into two-dimensional tables. The framework incorporates a wide range of typographic requirements for typesetting tables through extensions to the document style mechanism. Table layout is specified through grid designs similar to page layout grids familiar in the graphic arts. A mathematical linear inequality constraint solver is used to determine the final placement of table entries. A central idea in this approach is the separation of specifying the arrangement of table entries from computing their positions. Several interesting research problems arise from this framework. Some of those are outlined for future work in Chapter 6.

### 5.1.1 What do we need to do?

To support interactive table formatting, one needs efficient algorithms for WYSIWYG table display and for direct manipulation of both the table content and structure by pointing at and selecting components of the table. In an electronic document, tables can be represented by extending the idea of separating form from content, in this case by specifying the arrangement of table

entries separately from the content for each entry. Table entries can be arbitrary document objects such as text, illustrations, mathematical notations, or other tables. Typographic rules, decorations, and shaded backgrounds may be specified at the boundaries between table boxes. The data structure used here represents these boundaries explicitly, and thus supports all of these capabilities.

Abstractly, tables are two-dimensional rectangular arrangements of information that have both a row and column structure. Manipulating table content may require dealing with either a row or a column at different times. Thus the interactive operations should handle both row and column structures and should make it equally easy to select a row or a column and to perform a movement or editing operation on any selected table part. A selection hierarchy must permit identifying a distinguished table entry, a containing row or column of that table entry, a succession of enclosing rows or columns around the current selection, and ultimately the entire table. Operations on the selection hierarchy should work equally as well for groups of rows as for groups of columns. In particular, transposing the rows and columns of a table should be easy to accomplish with the table structure.

Style attributes for tables should be provided in a manner analogous to the way attributes were provided for illustrations in Chapter 3. The additional structure in tables poses some difficulties. Style attributes may be applied at several levels in a table. For instance, a typeface attribute might apply to the entire table, a single row within the table, a single column, a group of spanned rows or columns, or only to an individual table entry. A method is needed to determine the style attributes that apply to each table entry as it is formatted.

The table layout mechanism needs to handle not only general arrangements of rows and columns but also to specify the many alignments possible within a table design. Spanning a column heading across several columns is a special case of the general problem of aligning one set of entries with another set of entries. A robust framework for table formatting must be capable of expressing those arrangements and of determining the positions of the table boxes in such an arrangement.

A generalization of the table formatting problem permits overlapping layers of information. Background tints, such as the colored tints used in tables published in recent issues of the *Communications of the ACM*, form one class of overlapped information. This class of 2-1/2 dimensional overlapping can be handled without adding another dimension to the table arrangement algorithms.

### 5.1.2 How are we going to do it?

The framework for formatting tables that is proposed here has three parts: extensions to the document structure model, two-dimensional grid layout specifications (topology), and a layout constraint satisfaction algorithm (geometry).

The document structure and style machinery for tables is an extension of the model used in Chapter 3. Each table entry is a document object. Recall that a document object formats itself and is represented by a set of dimensions and two procedures, one for laying out the object and one for rendering the object. For tables, additional style attributes must be created for the new typographic features that appear only in tables, such as rule thicknesses and colors, alignments, and *bearoff distances* (the whitespace between table entries and grid lines). The tabular style attributes for a particular table entry may be collected into formatting style rules which are then attached as properties to the table entries. Since a table does not have the same simple hierarchy as the current tree-structured document model used in Tioga and extended for illustrations in Chapter 3, a more elaborate binding algorithm is needed to associate tabular style attributes with each entry.

The table arrangement, or *table topology*, is expressed using a grid design. The table layout, or *table geometry*, is computed from both the table topology and the dimensions of the table entries. A linear inequality constraint solver is used as a general mechanism for computing the table geometry.

Before discussing the table formatting system that has been implemented, it is appropriate to digress and look at some general strategies for two-dimensional layout problems. We would like to know how hard the problem of laying out tables is before tackling the solution. The general problem will be seen to be too difficult to solve in an interactive environment. Simplifying and restricting these problems leads to more realistic table formatting problems that can be handled interactively.

## 5.2 The Complexity of Table Formatting

Before discussing a framework for formatting tables, we examine some theoretical aspects of the problem to gain a perspective on its complexity. The problem of producing an optimal table layout seems intractable in its most general setting. The problem becomes tractable when sufficient structure is



added to the table arrangement. If a set of unordered table entries is to be arranged into minimum space, the problem is NP-complete. When only an approximation to the optimal size is needed (and the table can be formatted in horizontal slices) the problem is  $O(n \log n)$  where  $n$  is the number of table entries. When a total grid structure is imposed (entirely constraining the order in which entries are arranged into rows and columns) the table formatting problem becomes linear in the number of entries.

The general table formatting problem belongs to a class of two-dimensional packing problems first analyzed by Baker *et al.* [Baker, Packings]. The next subsection analyzes two versions of the table formatting problem that are NP-complete. We then look at restricted forms of the general problem which have simpler solutions. The simplest of these, GRID PACK, is then used as the basis for table formatting by grid structures, the main topic of this chapter.

### 5.2.1 RANDOM PACK

One type of table that occurs in practice is a completely unstructured collection of entries. A table containing a collection of photographs illustrating several aspects of a topic, in which the precise order of the photographs is unspecified, might require a formatting system to arrange the pictures to occupy the minimal space on a page. This can be formalized as RANDOM PACK, the problem of laying out an unordered set of rectangular table entries  $T = \{t_i\}$  with entry  $t_i$  having height  $h(t_i)$  and width  $w(t_i)$ , so that the resulting table is bounded by the page width and has a minimum depth (to be more precise, the RANDOM PACK problem asks whether the table can be laid out to fit on a page of depth  $k$ ). In laying out the table, whitespace may be left within the table boundary where entries do not abut exactly.

**THEOREM 1.** *RANDOM PACK is NP-complete.*

*Proof.* The proof follows from a reduction of the PARTITION problem, known to be NP-complete [Garey&Johnson, NP], to RANDOM PACK. Consider a table of rectangular entries, each with widths exactly one-half the page width but with arbitrary heights that sum to  $2k$ . The best possible table arrangement would be two columns of exactly the same height (in this case  $k$ ). This arrangement requires partitioning the entries into two subsets whose total height (the sum of the heights of its entries) is  $k$ . Thus, we are given a set of entries,  $T = \{t_i\}$  with heights  $h(t_i)$  such that  $\sum h(t_i) = 2k$ , and we wish to find two subsets of  $T$ ,  $S$  and  $S' = T - S$ , such that  $\sum h(s_i) = \sum h(s'_j) = k$ . This is precisely the PARTITION problem. There, we are given a set of integers and are

asked to partition them into two disjoint sets whose sum is the same. Given any instance of the PARTITION problem, we construct an instance of RANDOM PACK by specifying a set of rectangular table entries of width one-half the page and heights equal to the integers in the set to be partitioned. The original PARTITION problem has a solution if and only if the constructed RANDOM PACK table entries can be laid out with equally balanced column depths. This completes the reduction and the proof. ■

In fact, we can say more than this. Theorem 1 shows only that RANDOM PACK (and thus the general table layout problem) is NP-complete and hence almost certainly intractable. The PARTITION problem is known to have a dynamic programming algorithm that is polynomial time in the size of the set if there is a bound on the magnitude of the integers (the coefficients in the polynomial may be exponential in this magnitude) [Garey&Johnson, NP, page 90]. In formatting tables, the table entries are bounded by the page size, and thus the PARTITION problems that would be produced by the reduction in Theorem 1 would all have polynomial time algorithms. If the BIN PACKING problem is used instead of PARTITION we can in fact show that the table formatting problem is even more difficult than indicated by the the result of Theorem 1. BIN PACKING [Garey&Johnson, NP] is a problem which has no polynomial time algorithm unless  $P=NP$ , regardless of the magnitude of the numbers used in describing the problem (it is in the class of strongly NP-complete problems).

**THEOREM 2.** *RANDOM PACK is strongly NP-complete.*

*Proof.* We again perform a reduction from a problem known to be difficult. Consider a table of rectangular entries with integer widths less than the page width but with uniform heights. The table layout algorithm must minimize the number of rows required for the table in order to optimize space. This requires partitioning a set of table entries  $T=\{t_i\}$  having widths  $w(t_i)$  into  $k$  rows  $R_1, R_2, \dots, R_k$ . This partitioning is an instance of the BIN PACKING problem, known to be strongly NP-complete. The proof follows that for Theorem 1. ■

We realize that RANDOM PACK does not produce very interesting or aesthetically pleasing tables. Few tables contain unordered entries (although the example given before, a collection of related but unordered photographs, is such a table). Even when a table is an unordered set of entries, the mix and match jumble produced by RANDOM PACK is not typical of the row and column structure usually employed for such tables. The juxtaposition of different sizes and shapes for the various entries does not lend itself to a readable table. An aesthetically pleasing table exhibits a design discipline for organizing the table entries that avoids unexpected relationships due to the random positioning of

table entries. We are thus led from this general problem to introduce restrictions that make the problem more realistic and also more tractable for formatting, especially for interactive work.

### 5.2.2 STUB PACK

There are two approaches to dealing with the general table formatting problem: accept an approximate solution to the optimal table layout or restrict the table formatting problem so that efficient optimal algorithms exist. Approximation algorithms for the TWO-DIMENSIONAL BIN PACKING problem, and hence for the RANDOM PACK problem, are known [Coffman, 2D Packing]. These approximations use polynomial time algorithms to produce layouts within  $5/4$  of the optimal space [Baker,  $5/4$  Algorithm]. The Up-Down (UD) algorithm presented by Baker *et al.* is known to have a bound  $UD(L)$  on the packing height for any list  $L$  of rectangles with height at most  $H$  as follows:

$$UD(L) \leq 5/4 \cdot OPT(L) + 53/8 \cdot H,$$

where  $OPT(L)$  is the height of the optimal packing of the list  $L$  of rectangles (as would be produced by RANDOM PACK).

However, the table layouts produced by these algorithms are still unaesthetic, with a chaotic arrangement of small entries among the crevices between larger ones, as shown in the example layout of Figure 5-1 adapted from Figure 2 in Baker *et al.* [Baker, Packings].

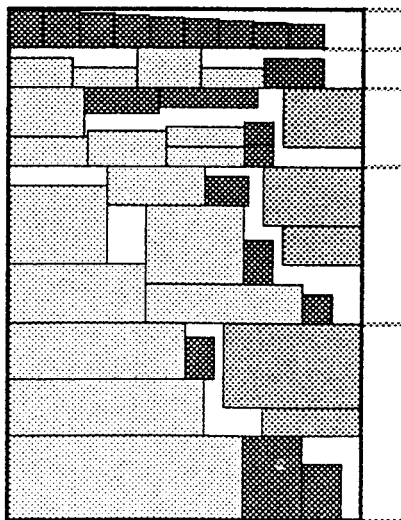


Figure 5-1. AN APPROXIMATELY OPTIMAL LAYOUT produced by the Up-Down packing algorithm due to Baker *et al.* [Baker, Packings]. The example is similar to Figure 2 in their paper. The light grey, medium grey, and dark grey boxes were packed with three different strategies.

Restricting the table arrangement leads to more aesthetic table layouts and to algorithms with polynomial running time. Introducing a row structure to the table by placing horizontal rules between the rows is the first restriction we consider. Once a rule is introduced into the table, it must extend all the way to the right of the table. These rules form the stub of the table (typical of financial tables) and hence this restricted table formatting problem will be called STUB PACK. Figure 5-2 contains an example of a STUB PACK table layout.

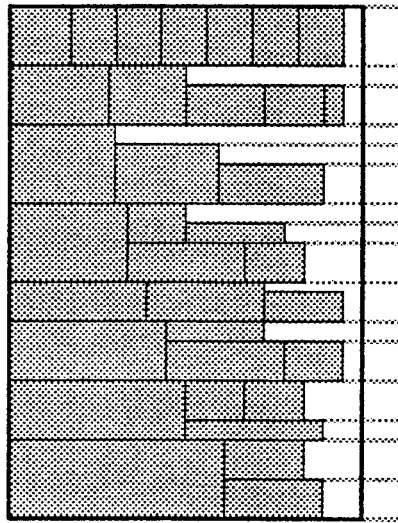


Figure 5-2. A HYPOTHETICAL STUB PACK table layout with the largest elements at the bottom left and the smaller elements packed from left to right.

STUB PACK can be formalized as a set of unordered table entries  $T = \{t_i\}$  with height  $h(t_i)$  and width  $w(t_i)$  that are to be placed into an arrangement bounded by the page width and having minimum depth. Each entry in the final layout is separated from the entry below it by a line (typographic rule) that defines a row of the table.

A polynomial time algorithm exists to solve the STUB PACK problem within 1.7 of the optimal space [Coffman, 2D Packing, Theorem 2]. The First-Fit Decreasing-Height (FFDH) algorithm described by Coffman *et al.* is shown to have a bound on the packing height of

$$\text{FFDH}(L) \leq 1.7 \cdot \text{OPT}(L) + 1.$$

The FFDH algorithm takes a list  $L$  of rectangles with arbitrary widths ordered by nonincreasing height and places each rectangle left-justified on the first level in which it will fit, or if none of the levels will accommodate the rectangle, a new level is begun. The FFDH algorithm requires  $O(n^2)$  time for  $n$  rectangles

in the list. A linear post-processing pass may be added to improve the aesthetics of the table by distributing the excess whitespace within a row and around each entry.

While STUB PACK has a polynomial time algorithm, it still does not lay out the ordered tables normally encountered in practice. The row and column grid structure that occurs in tables imposes an ordering of table entries specified by the table designer, who has taken into account a number of aesthetic considerations. Restricting tables to be ordered in advance by the designer reduces the complexity of the table formatting problem (at the expense of possibly requiring greater space). An algorithm which performs table layout given the arrangement is what we need to achieve the aesthetics we desire.

### 5.2.3 GRID PACK

The GRID PACK problem is to lay out a given set of table entries, each with a width and a height, where all of the entries are assigned to lie between particular row and column grid coordinates within the table. The layout is entirely determined once the physical (page) coordinates of the grid lines are known. These are determined by the width and height of the entries.

*THEOREM 3: GRID PACK requires linear time.*

*Proof:* The requirement is to determine the position of the grid lines. This can be done by a two pass algorithm that examines all of the table entries in turn. The first pass determines the relative width and depth of each row and column (horizontal and vertical pairs of grid lines) by ensuring that the relative width of the column is greater than the width of any entry in that column and that the relative depth of the row is greater than the height of any entry in that row. The absolute page coordinates of the row and column grid lines are then determined by accumulating relative widths and depths for consecutive grid lines. The second pass over the entries determines the page coordinates of each entry from the row and column grid coordinates. ■

The GRID PACK algorithm is similar to the one used by the `tbl` table formatter. Additional alignment possibilities may be incorporated into the GRID PACK problem through suitable extensions. As stated here, it does not handle spanned column headings or aligning table entries on decimal points. The remainder of this chapter will discuss the practical issues of table formatting, including a document structure and grid system for representing table layouts and an interactive algorithm for formatting these tables.

## 5.3 Table Document Structure

The document structure outlined in Chapter 3 for including graphical illustrations in documents is a tree-structured hierarchy of document objects. The same object-oriented strategy can be used to extend the document structure to represent tables by adding another class of document content objects. The recursive document structure still pertains, and text documents may contain tables that in turn contain text, illustrations, other tables, or any other document object that has been defined. Unlike the pipelined structure of `troff` preprocessors, this recursive structure implies no ordering or ranking among the document object classes, and the recursion can start with any object. This permits a general and extensible treatment of information presented within documents, illustrations, and tables. An example of a table with varied content is shown in Figure 5-3.

### 5.3.1 Table Entry Representation

For formatting purposes, a table object in the extensible document structure can be thought of as a box and two procedures for laying out and rendering its content. The box concept is similar to the boxes used in `eqn` and `TEX` but with additional information required for alignment of table entries. The procedure concept is similar to the artist procedures used to render data structures graphically [Myers, Incense] or animate graphical objects [Kahn&Hewitt, Actors].

A table entry is abstractly represented by a box with four offsets (left, right, up, and down) from an origin implied by the content. This origin is the starting point for the rendering procedure that displays the content of the table entry. Four offsets, rather than simply a width and a height, permit alignment in both the horizontal and vertical directions simultaneously. Other schemes like `eqn` and `TEX` that represent boxes with fewer parameters provide less-functional alignment primitives. To align the table entry with other entries an alignment point is computed within the box depending on the typographic alignment style, as shown in Figure 5-4.



	<p>Chester Carlson</p>	<p>Formatted Document</p> <table border="1" data-bbox="987 548 1320 940"> <tr> <td>Chapter 3</td> <td>12</td> </tr> <tr> <td colspan="2">A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.</td> </tr> <tr> <td colspan="2">Section Heading</td> </tr> <tr> <td colspan="2">A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.</td> </tr> </table>	Chapter 3	12	A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.		Section Heading		A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.	
Chapter 3	12									
A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.										
Section Heading										
A paragraph of text goes here to demonstrate a thumbnail sketch of a formatted page.										
 <p>A Rose is a Rose</p>	<p>Griffin Rose</p>	<p>Table</p> <table border="1" data-bbox="886 1003 1308 1129"> <tr> <td>0</td> <td><i>speed</i> × <i>time</i></td> </tr> <tr> <td>.625</td> <td><i>acceleration</i> × <i>time</i></td> </tr> <tr> <td>1023.5</td> <td><i>force</i> × <i>distance</i></td> </tr> </table>	0	<i>speed</i> × <i>time</i>	.625	<i>acceleration</i> × <i>time</i>	1023.5	<i>force</i> × <i>distance</i>		
0	<i>speed</i> × <i>time</i>									
.625	<i>acceleration</i> × <i>time</i>									
1023.5	<i>force</i> × <i>distance</i>									

Figure 5-3. A WIDE RANGE OF CONTENT can be incorporated within tables using an object-oriented document structure. This table includes five different kinds of content: text, a scanned illustration, a synthetic computer generated drawing, composed pages, and another table. The text captions in the center column are positioned flush at the top of each row and alternate flush right and left. The picture of Chester Carlson, the inventor of xerography, was scanned from an original photograph and is 367 scan lines by 474 pixels with each pixel containing an 8-bit grey value. The formatted document is the output of other software that produces a compatible printer format used at Xerox PARC. The synthetic graphic image was created by Maureen Stone with the Griffin illustrator. Both the table and the subtable were composed using the table formatting prototype described in this chapter.

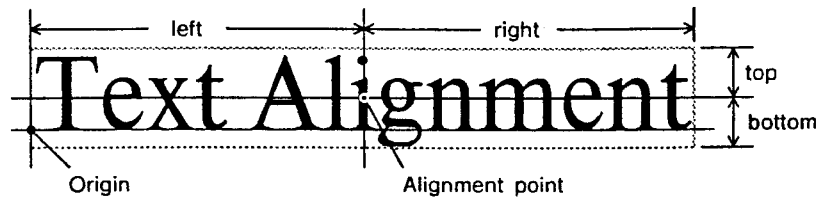


Figure 5-4. A TABLE ENTRY BOX is represented by four offsets that are the left, right, up, and down distances from an alignment point. These offsets are computed from the dimensions of the box and the typographic alignment attributes, in this example, centered both horizontally and vertically.

The layout procedure is given the arrangement and content of the table entries and determines their sizes and positions. A table's arrangement is explicitly determined by its creator when the table entries are collected in the document. This is analogous to the ordering of paragraphs in a document as it is created by an author.

The rendering procedure is given the positions of the table entries, as computed by the layout procedure, and produces the human-readable form of the table by recursively rendering the content of the table entries at each position. The rendering procedure relies on a device-independent graphics package that can produce a screen display or a file-based printer description of the table. This device-independent capability permits WYSIWYG interaction with table objects in their actual positions, subject to differences in device resolution. The separation of the layout and rendering steps permits the rapid redisplay of the table without recomputing the layout when the view of a table is moved or scrolled.

### 5.3.2 Table Arrangement

The document structure extensions for tables must capture the arrangement of the table entries as well as their content. Tables are two-dimensional, and in this respect are similar to mathematical notation in the microscopic sense, and to page layout in the macroscopic sense. However, tables are significantly unlike paragraphs of text, wherein text flows from one line to the next and there are few positioning relationships between text elements. It is precisely the positioning and alignment relationships between table entries that must be included in the table document model.

One might begin with the obvious row and column structure of the table. Many document formatting systems have chosen one of these as the dominant structure and expressed the table arrangement in terms of it, leaving the other



structure implicit. For example, rows are the dominant structural element for tables in `tbl`. This choice appears natural since all the column entries of a row can be entered on the same input line, and the UNIX philosophy treats streams of characters as the unifying data structure.

This row dominance makes column operations more difficult. Adding a new row is simple; one adds a new input line to the source file. Adding a new column is tedious; each input line must have a new column entry inserted in the appropriate place, although special editors could help here. Another asymmetry between rows and columns in `tbl` concerns how one specifies spanned headings. A spanned column heading must be specified as a sequence of `s` layout codes for each column spanned, whereas a spanned row heading may be specified either by a `↑` layout code for each row spanned or by a special formatting code `\↑` as the table entry content for each spanned row entry. The treatment of rows and columns in interactive table formatting should be symmetric to reduce the cognitive burden and to avoid errors.

A hierarchical structure suggests itself from the subdivision of rows and columns in tables. Figure 5-5 illustrates a table with its row and column subdivisions and the hierarchical data structure based on the columns. A column heading that spans several entries becomes a subtree root for those columns. Multiway branches are for spanned headings, single branches are for table entries. A dual hierarchy would be expected for the row structure. Such a dual hierarchical scheme was unsuccessfully attempted in a previous table formatting prototype [Beach, CS740 project]. The implementation was complex and the data structures were difficult to coordinate. Many of the algorithms had to be duplicated for the row and column cases. The crucial realization was that not all table designs can be represented as hierarchical tree structures. In particular, the table in Figure 5-5 does not have a hierarchical row structure because heading D has two row parents, C and G. A more general underlying structure is required. This is the topic of the next subsection.

When they exist, the dual row and column hierarchies do provide powerful interactive selections of parts of tables. This selection hierarchy notion is similar to a text selection hierarchy which enables one to select a character and extend the selection through various levels of structure from a character to a word, sentence, paragraph, section, and ultimately the entire document. Starting from a selected table entry, one can extend the selection to include all entries in the containing row or column by traversing either the row or column hierarchy. Successive extensions of the selection would include containing rows or columns until the entire table is selected. While a hierarchical table representation was

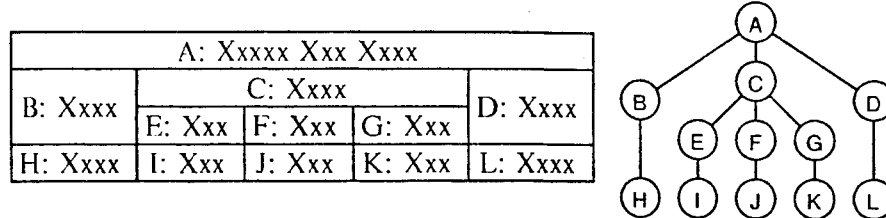


Figure 5-5. HIERARCHICAL TABLES contain rows and columns that subdivide into other rows and columns like parent and children nodes in a tree. The column structure of the table on the left is shown in the graph on the right. All the table entries are labelled to show the correspondence. Note that this particular table does not have a row hierarchy because entry D has two row parents, C and G.

not used in table formatting prototype, the selection hierarchy serves as a good model for manipulating tables and was implemented for the grid representation described next.

### 5.3.3 Grid Structure

A more general representation of table arrangements must simultaneously satisfy several goals: it must exist within a hierarchical document model, it must embed arbitrary table entry objects, and it must describe the arrangement and positioning relationships of the table entries. The hierarchical document model cannot be used directly because tables are not always hierarchical, for example, the box head in Figure 5-5 is nonhierarchical. The insight applied here is that one need provide the data structure for tables explicitly in the document model. Instead, the table arrangement can be described indirectly in terms of a grid coordinate system with table entries having associated grid coordinates. This permits a table object to be represented by a collection of arbitrary table entry objects and the table arrangement to be specified by the grid labelling. The entries may be listed in any order since the labelling will determine the table layout.

The proposed grid coordinate scheme expresses the table topology. This in turn determines the arrangement of table entries between the grid lines, and thus determines which table entries are neighbors and which entries share the same boundaries. Grid modules between grid lines may be nonuniformly spaced to permit both narrow and wide entries. Each table entry is surrounded by four grid lines, one each on its left, right, top and bottom sides. A row is bounded by two horizontal grid lines, and a column by two vertical grid lines in a symmetric fashion. Entries in the same row will share the same pair of

horizontal grid lines, similarly for entries in the same column. A spanned column heading will cross several grid lines and is bounded by the left grid line of the leftmost spanned entry and by the right grid line of the rightmost spanned entry. An arbitrarily complex arrangement of table entries can be specified by listing the four grid lines for each table entry. Figure 5-6 contains a table with the grid coordinate system overlaid as light grey lines.

	Spanned over Four Columns			
	Equal Width		Unequal Width	
	Very Wide	Thin	Very Wide	Thin
First Row	XXXXXXXXXX	XXX	XXXXXXXXXX	XXX
Second Row	XXXXXXXXXX	XXX	XXXXXXXXXX	XXX

Figure 5-6. TABLE WITH GRID COORDINATE SYSTEM in which the grid lines are overlaid in light grey. Some table entries are contained within a single grid module while others span across several grid modules. The two black typographic rules, one horizontal below the column headings and one vertical after the row stub, run along the grid boundaries.

Typographic rules and decorations can be superimposed on the grid boundaries. In this case, the grid boundary has a nonzero width equal to the thickest rule that runs along that grid line. Note that both horizontal and vertical rules are treated symmetrically.

The table layout geometry, that is, the actual position of each grid boundary and each table entry, is computed from both the table topology and the dimensions of the table entry contents. A mathematical linear inequality constraint solver permits general alignment relationships to be expressed, such as the equal column widths shown in Figure 5-6. The independent variables in the inequality constraints are the positions of the grid lines and the alignment points of table entries. An incremental constraint solver that can accept new or changed constraints permits interactive modification of the table arrangement.

### 5.3.4 Graphic Arts References to Grid Systems

Similar grids have been used in the graphic arts since grid systems were first developed by designers in Switzerland shortly after World War II. The principle behind a grid system is an objective attitude to the presentation of the subject material and to the uniformity in the layout of all pages. Grids institute a disciplined approach to design and layout, limiting the number of choices to provide regularity and order in potentially chaotic situations.

“The fewer the differences in the size of the illustrations, the quieter the impression created by the design.” [Müller-Brockman, *Grid Systems*, p11]

Several graphic designers have written about grid systems in books including Müller-Brockman's *Grid Systems in Graphic Design* [Müller-Brockman, *Grid Systems*], Hurlburt's *The Grid* [Hurlburt, *The Grid*], and Williamson's *Methods of Book Design* [Williamson, *Book Design*]. Some computer composition systems have incorporated grid systems. One early example was Tilbrook's NEWSWHOLE [Tilbrook], a prototype system for interactive newspaper page layout. Hurlburt [Hurlburt, *The Grid*] describes the modular design and sense of proportion induced by a grid design. Several proportions, such as the Golden Ratio, square, and  $\sqrt{2}$  rectangle, are commonly used as the basis for grid designs. Letter form design is also often constructed to conform to a grid design [Goines, *Constructed Alphabet*].

Unlike the orthodox grid formed by uniformly spaced horizontal and vertical lines to produce square modules, the typographic grid is more casual with lines introduced to specify margins on the page, column measures, and alignment guides. Common grid lines are the headline, the first line of text on a page, the first line of a chapter title, the first line of text within a chapter, the last text line on a page, and the footline. Figure 5-7 shows the grid design for the pages of this thesis.

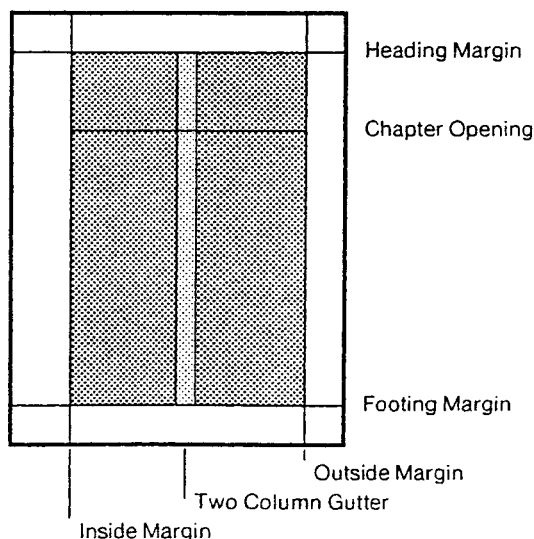


Figure 5-7. GRID DESIGN for the pages of this thesis illustrates the traditional use of grid boundary lines to determine margins, column measures, gutter widths, alignment points, etc. The grid combines both a single column format (the large grey rectangle) for text pages and a double column format (the two dark grey rectangles) for glossary and index pages.

The interactive newspaper pagination system NEWSWHOLE [Tilbrook] is an example of the application of grid systems for layout and formatting. The grid lines in that system were active; they could be stretched and moved across the page at the user's command. There was a simple constraint satisfaction to ensure that there were always an integral number of grids across the section of the page and that the grids were equal width. The NEWSWHOLE prototype had only limited text composition capabilities, and it did not attempt a WYSIWYG presentation of the entire page (mainly due to display resolution, which will always be a problem). The same idea can be used here to do interactive tabular composition and is discussed in Chapter 6.

### 5.3.5 Style Attributes for Tables

The document style mechanism must also be extended to incorporate additional table formatting attributes and to apply those attributes to table entries. The additional tabular style attributes include various alignment specifications (decimal or character alignment, top or bottom baseline alignment, centered top or bottom baseline alignment), various rule parameters (rule thickness and color), and various bearoff distances (the whitespace between table entries and the grid lines).

Applying style attributes to table entries is harder than for text or illustrations because the attributes may come from several nonhierarchical sources. Figure 5-8 illustrates some of the possible interactions among style attributes applied to parts of a table. The whole table may have a general formatting style. Rows or columns may have particular style rules for the entries within those rows or columns. Each table entry may have additional local formatting attributes applied that override the other specifications.

Because tables have both a row and a column structure that in turn has nested subrows and subcolumns, there may be several row or column style rules applicable for a given table entry. The nested containment of a table entry within a row and/or a column can determine an appropriate search path for applying style rules. The rendering algorithm first applies the style attributes specified in the style rule for the entire table. It then applies attributes for each containing row or column that successively contains the table entry, down to the attributes for the given table entry. When style rules do not specify values for all the possible style attributes, values are inherited from previous style rules along the search path. Style rules may be specified for both a row and a column containing the given table entry, in which case it is necessary to disambiguate the row or column preference. The solution taken here is to provide a Boolean

Stub Head	Spanning Column Head				
	Col Head	Col Head			Col Head
		Col	Col	Col	
Row	value	<i>value</i>	<i>value</i>	<i>value</i>	value
Row	<b>value</b>	<b><i>value</i></b>	<b><i>value</i></b>	<b><i>value</i></b>	<b>value</b>
Row	value	<i>value</i>	<i>value</i>	<i>value</i>	value

Figure 5-8. STYLE ATTRIBUTES for a table entry are determined by several style rules specified for the entire table, a row, a column or an individual table entry. This table style rule specifies a Helvetica type family. One row has a style attribute for bold face. The spanned column has a style attribute for italic face. One of the three table entries in the intersection of the bold row and italic column has a Times Roman type family attribute, which overrides the global specification. The style attributes for a particular entry are determined by accumulating all the style attributes according to a natural search order: table, row or column (according to a preference choice), then table entry.

selector as a property of the table to choose between row-before-column or column-before-row preferences.

Alignment relationships between table entries are controlled by the alignment points of the table entry objects. These relationships are generalizations of the `mark` and `lineup` alignment specifications for equations in `eqn`. Formatting style attributes determine the alignment point based on the typographic treatment of the table entry. Centered alignment places the alignment point in the middle of the object box by dividing the sum of the appropriate pair of dimensions by two. Flush left, right, top or bottom alignment places the alignment point at the appropriate edge of the box. Aligning on a character within the table entry sets the alignment point to be the origin (which may be chosen from one of several alignment options) of that particular character. Figure 5-9 illustrates two columns of character-aligned table entries, the first aligned on decimal points (including the implied decimal point at the end of a numerical string), and the second aligned on a multiplication sign.

0	<i>speed × time</i>
.625	<i>acceleration × time</i>
1023.5	<i>force × distance</i>

Figure 5-9. ALIGNMENT ON A CHARACTER within a table entry may be based on specific characters within lines of text, such as decimal points (actual or implied) in the first column, and multiplication signs in the second column.

The positioning of odd-sized table entries requires additional information, especially for text entries that are folded into multiple lines or for entries with varying heights like mathematical expressions. The top row in Figure 5-10, which has been horizontally aligned on the center of each table entry, demonstrates the disjointed result when baselines of folded entries do not align across the column. Simple horizontal alignment attributes such as flush top, flush bottom, or center do not take into consideration the internal structure of a table entry. Additional alignment choices are needed to align on the baselines within a text table entry, such as align on the top or bottom baseline. Text entries with an even number of baselines have two middle baselines for centering, thus requiring two choices to center on either the top-middle or bottom-middle baseline. The bottom row of Figure 5-10 demonstrates the range of alignment choices available for aligning baselines within table entries.

Horizontally Centered, Ignore Baseline	<u>Baseline</u> Baseline	Baseline <u>Baseline</u> Baseline	Baseline Baseline <u>Baseline</u> Baseline	Baseline Baseline <u>Baseline</u> Baseline Baseline
Horizontally Centered, Ignore Baseline	<u>Baseline</u> at Top	Bottom <u>Baseline</u>	Center <u>Baseline</u> at Top of Many	Center on Bottom <u>Baseline</u> of Many

Figure 5-10. HORIZONTAL ALIGNMENT OF FOLDED TABLE ENTRIES may not produce aesthetic results when the entry has several lines of text. All of the entries in the top row are horizontally aligned without regard to their internal structure and the baselines do not align; all of the entries in the bottom row have been centered on a baseline indicated by the text of the entry.

Establishing the alignment point on baselines within a table entry implies that the document object must understand all these alignment choices. The simpler alignment attributes, such as flush top or bottom, could be computed from the representation of an object and its four offsets (except for decimal or character alignment). Some table entry objects have little similarity to text objects, yet the alignment attributes must apply to all objects in a uniform way. The technique used in the prototype is to extend the representation of a table

entry to include an optional list of baseline origins. Should the list be absent, the default baseline is chosen to pass through the origin of the object. Alignment attributes that concern themselves with content, such as decimal alignment, are left as the responsibility of the table entry object. Some objects may choose to ignore text alignment attributes. For example, a scanned illustration object contains no characters, so it could safely ignore those attributes.

Another alignment attribute deals with the excess whitespace in the wide columns in Figure 5-11. In this example, the table entries in each column are aligned on decimal points. The last two columns need further specification: how should the excess whitespace induced by the very long column heading be treated? This additional specification determines how the vertically aligned set of column entries are positioned within the column. The choices are centered, flush left, or flush right. Currently, the table formatting prototype does not support this positioning specification, but a more complete implementation is planned and described in Chapter 6. In `tbl`, numeric entries that are aligned on decimal points are centered within the column after they are aligned on decimal points; there is no control provided over the distribution of the excess whitespace.

---

Short Head			Very Long Column Head Over Narrow Entries	
54.321	54.321	54.321	54.321	54.321
654.32	654.32	654.32	654.32	654.32
54.321	54.321	54.321	54.321	54.321

---

Figure 5-11. SPECIFYING POSITION WITHIN A COLUMN as well as aligning table entries may be necessary when there is excess whitespace to disperse among the row or column entries. All the numeric table entries are aligned on decimal points. The last two columns have excess whitespace due to the very long column head; the first set of aligned column entries is positioned flush right within the column and the second is flush left. Similar specifications are necessary for rows also.

---

### 5.3.6 Text within Table Entries

Textual table entries may be more complex than just simple phrases or numbers. Some entries may be paragraphs with long lines of text broken into pieces short enough to fit within the column boundaries. Yet the layout algorithm attempts to determine the width of the column boundaries from the widest line in the broken paragraph. Thus the line-breaking algorithm depends on the column width and the column width depends on the width of lines



folded by the line-breaking algorithm. How does one exit from this circular dependency?

In existing table formatters, such as `tbl`, the only technique for dealing with this problem is to require the table designer to supply an explicit column width attribute. In fact, this is an easy way to specify equal-width columns, since no matter what the content of the entry, all such designated column entries are forced to be the specified width. Unfortunately, fixing the column width often results in an unaesthetic table. Short justified lines induce more hyphenation and larger spaces between words, making the entries more difficult to read. Unjustified lines are easier to read but extra whitespace accumulates at the end of lines creating the illusion of unevenly spaced columns.

A better strategy would be for the formatter to accept the line length as only a 'hint' for folding the text to approximately that length. The actual bounding box of the composed text could be returned by the object layout procedure or determined by a scan of the formatted object. With the bounding box, a folded table entry can be treated uniformly as any other entry. A feature of this strategy is that a column of folded entries is only as wide as necessary without excess whitespace.

The problem of determining an aesthetically pleasing column width for a column of entries with widely varying widths or for a table that exceeds the page measure is a very hard problem and is not solved here. Some directions for determining optimum column widths for folded table entries and for balancing whitespace are outlined in Chapter 6.

## 5.4 Implementation of the Tabular Grid Structure

We will need two table representations, one for capturing a table in an external representation of a document and another internal data structure for interactive manipulation. This section describes both representations used in the prototype table formatter that has been implemented in the Cedar programming environment.

### 5.4.1 External Representation of a Table in a Tioga Document

An external storage representation is normally concerned with compactness and ease of conversion between external and internal representations. However, because the prototype relies on the Tioga document structure, the external

representation of a table must be a textual description of the table. This representation is editable by the Tioga text editor. While this is an advantage for testing the prototype, it is expected that an interactive user interface will be built to permit most editing actions without resorting to editing the textual representation directly. We believe, however, that the textual description is useful for debugging, for document interchange, and for possibly fine tuning table structures.

A table is represented as a subtree of nodes in a Tioga document hierarchy. The root node of the table subtree has an `ArtworkClass` property (described in Chapter 3) of `Table` to distinguish the subtree as being a table object (as distinct from text or illustrations). This `Table` property is recognized by the table formatting prototype which interprets the contents of the subtree as a collection of table entries. The table entries themselves may contain an object of any document object class since the object layout and rendering procedures understand the content of the object. This object-oriented design permits arbitrary recursion within document content at any time.

The table representation is converted into the internal data structure whenever a table is edited in a WYSIWYG fashion or formatted. After manipulating the internal data structure (perhaps changing the table topology or the content of some table entries) the internal data structure is converted back into a Tioga document subtree. The new subtree is spliced into the original document to replace the previous table representation.

Only topological information is represented in the table description. Table entries are described by the grid coordinates that they occupy. Typographic rules are described by the grid lines along which they run. The table previously shown as Figure 5-6 has been annotated with grid coordinate values in Figure 5-12. Horizontal and vertical grid lines are each numbered beginning at 0 and incrementing for each successive grid line. A column or row is contained within two grid lines (not necessarily two consecutive grid lines). Thus the column containing the heading "Spanned over Four Columns" is contained within vertical grid lines 1 and 5. Two typographic rules are shown along vertical grid line 1 and along horizontal grid line 3. The table in Figure 5-12 will be used both to illustrate the external table representation (in Figure 5-13) and the internal data structure (in Figure 5-15).

	0	1	2	3	4	5
0		Spanned over Four Columns				
1		Equal Width		Unequal Width		
2		Very Wide	Thin	Very Wide	Thin	
3	First Row	XXXXXXXXXX	XXX	XXXXXXXXXX	XXX	
4	Second Row	XXXXXXXXXX	XXX	XXXXXXXXXX	XXX	
5						

Figure 5-12. A TABLE WITH A GRID COORDINATE SYSTEM superimposed. The table is a duplicate of Figure 5-6 that will be used to illustrate the external Tioga document structure in Figure 5-13, and the internal corner stitched data structure in Figure 5-15.

A table is described in general terms by the contents of the root node of its table subtree. In Figure 5-13, the table is formatted on a 6 by 6 grid (for 5 rows and 5 columns). A grid overlay 3-points thick in a light gray color (hue 0, saturation 0, brightness 0.7) is superimposed on the table to make the grid coordinate system visible. (This grid overlay would not appear on printed tables, except for expository purposes, although the grid overlay serves as a useful target for interactively manipulating the grid topology.) The direct descendants of the table root node within the document describe the table structure and table entries in more detail. There are four things to describe: constraints on the grid coordinate layout, boxes containing table entries, typographic rules, and background shaded areas.

### *Constraints*

Constraints are linear inequalities or equalities generated automatically by the table layout procedure. The table in Figure 5-12 has two auxiliary constraints supplied by the table designer to force equal column widths. The first constrains the column between grid lines 1 and 2 to have the same width as the column between 2 and 3. The second constraint forces the column between grid line 0 and 1 to have the same width as the column between 1 and 3 (a column which spans two grid lines). The complete details of the constraint mechanism are deferred until Section 5.5.

---

```

Grid 5 Rows 5 Columns GridOverlay 3 pt 0 0 0.7
  Constraint 2*C2 - 1*C1 - 1*C3 = 0
  Constraint 2*C1 - 1*C0 - 1*C3 = 0
  Box (0,1) (1,5) TopBaseline Center
    Spanned over Four Columns
  Box (1,1) (2,3) TopBaseline Center
    Equal Width
  Box (1,3) (2,5) TopBaseline Center
    Unequal Width
  Box (2,1) (3,2) TopBaseline Center
    Very Wide
  Box (2,2) (3,3) TopBaseline Center
    Thin
  Box (2,3) (3,4) TopBaseline Center
    Very Wide
  Box (2,4) (3,5) TopBaseline Center
    Thin
  Box (3,0) (4,1) TopBaseline FlushLeft
    First Row
  Box (3,1) (4,2) TopBaseline Center
    xxxxxxxxxx
  Box (3,2) (4,3) TopBaseline Center
    xxx
  Box (3,3) (4,4) TopBaseline Center
    xxxxxxxxxx
  Box (3,4) (4,5) TopBaseline Center
    xxx
  Box (4,0) (5,1) TopBaseline FlushLeft
    Second Row
  Box (4,1) (5,2) TopBaseline Center
    xxxxxxxxxx
  Box (4,2) (5,3) TopBaseline Center
    xxx
  Box (4,3) (5,4) TopBaseline Center
    xxxxxxxxxx
  Box (4,4) (5,5) TopBaseline Center
    xxx
  Rule (0,1) (5,1) 1 bp
  Rule (3,0) (3,5) 1 bp

```

Figure 5-13. A TABLE OBJECT IN THE EXTERNAL REPRESENTATION of Figure 5-6. Each line corresponds to a document node with indentation depicting the nesting relationship of content nodes within table entry nodes. The first node is the table subtree root and describes the table in general. Children of the table root describe constraints on table alignment, table entries, typographic rules, and backgrounds. Children of table entries are the content of the table entry.

---

### *Table Boxes*

Table entry boxes are specified by their top-left and bottom-right grid coordinate pairs. The content of the table entry is specified as the descendant of the table entry node. This descendant node may be any document object.

The first table entry in Figure 5-13 describes the spanned heading. Its coordinates are from (0, 1) to (1, 5), spanning four grid modules. The alignment attributes are `TopBaseline` (horizontal alignment) and `Center` (vertical alignment). The object within this table entry is the text object, descended from the first table entry and shown indented in the figure, containing the phrase "Spanned over Four Columns". The next table entry also spans column entries, has the same alignment attributes, and contains the text object "Equal Width".

Table entries may appear in the document representation in any order because the coordinate specifications determine where in the table they will appear. The nesting of objects as descendents of their corresponding table entry description is the only requirement for including content within a table representation.

### *Typographic Rules*

Typographic rules run along grid lines and have a certain specified thickness. The two rules in Figure 5-12 are both 1-point thick. One rule runs along vertical grid line 1, through the entire table, and the other rule runs along horizontal grid line 3. Rules may run along portions of a grid line. Thus, a vertical rule under the spanning head to separate the equal and unequal pairs of columns might run from grid coordinate (1, 3) to (5, 3).

### *Backgrounds*

Background areas are specified by the rectangle they shade and the color of the shading. The rectangle is specified by two pairs of coordinate values just as for table boxes. The color specification could be any recognized color naming scheme, and in the prototype, color is given as hue, saturation, and brightness.

## 5.4.2 Corner Stitching Data Structure

The internal data structure for representing tables must meet several criteria. It must represent the arrangement of table entries in both the horizontal and vertical directions. It must support row and column selections and a selection

hierarchy within containing rows and columns. It must also permit fast algorithms suitable for interactive table editing operations and for conversion to/from an external form.

The data structure chosen for the table formatting prototype was the corner stitching data structure employed by Ousterhout [Ousterhout, Corner Stitching] for his VLSI layout tools. An implementation by Shand [Shand, CornerStitching] which runs in the Cedar environment was modified and extended for the prototype.

The corner stitching data structure tessellates the plane with rectangular tiles. Each tile is connected to its neighbors by four compass-point links and each tile references its associated data. The corner stitching implementation uses special border tiles surrounding the tessellation to simplify the algorithms. Figure 5-14 illustrates a prototypical tile and a fragment of the tessellation showing how several tiles are connected together.

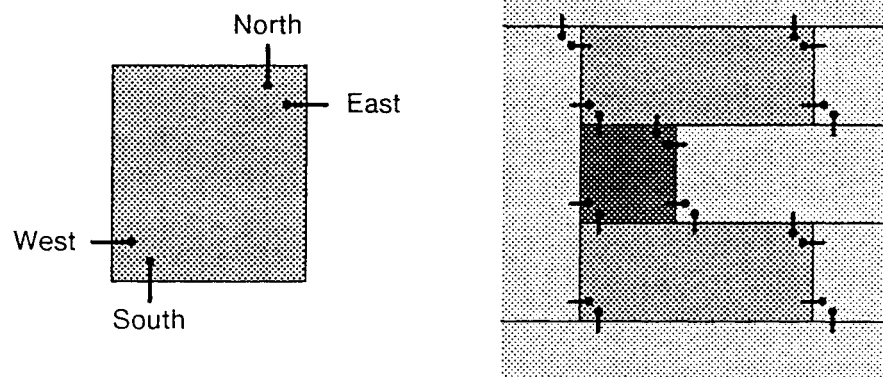


Figure 5-14. CORNER STITCHED DATA STRUCTURE uses tiles that are joined together by four pointers, two at the NorthEast corner going North and East, and two at the SouthWest corner going South and West. The example on the right shows three tiles stitched together surrounded by special border tiles, shown in light grey, that are used to simplify the algorithms.

To use the corner stitching data structure there must be a geometric coordinate system. The table topology specified by the grid structure has no geometric information, only the presence or absence of table entries and their relative positioning. Grid boundaries between table entries must have nonzero width, even in the table topology, since provision must be made for typographic rules. Therefore the corner stitching coordinate system has distinct coordinate values for table entries within the grid lines and for typographic rules along the grid lines. The topology is highlighted in the table prototype by representing table entries as unit squares and grid lines as lines with unit widths. (Some additional separation between grid lines and table entries is provided in the

prototype to help distinguish between adjacent tiles. Thus the table grid coordinates are in fact multiplied by four to create corner stitching coordinate values.) The coordinate mapping is indicated in Figure 5-15, which presents the corner stitched data structure for the table in Figure 5-6.

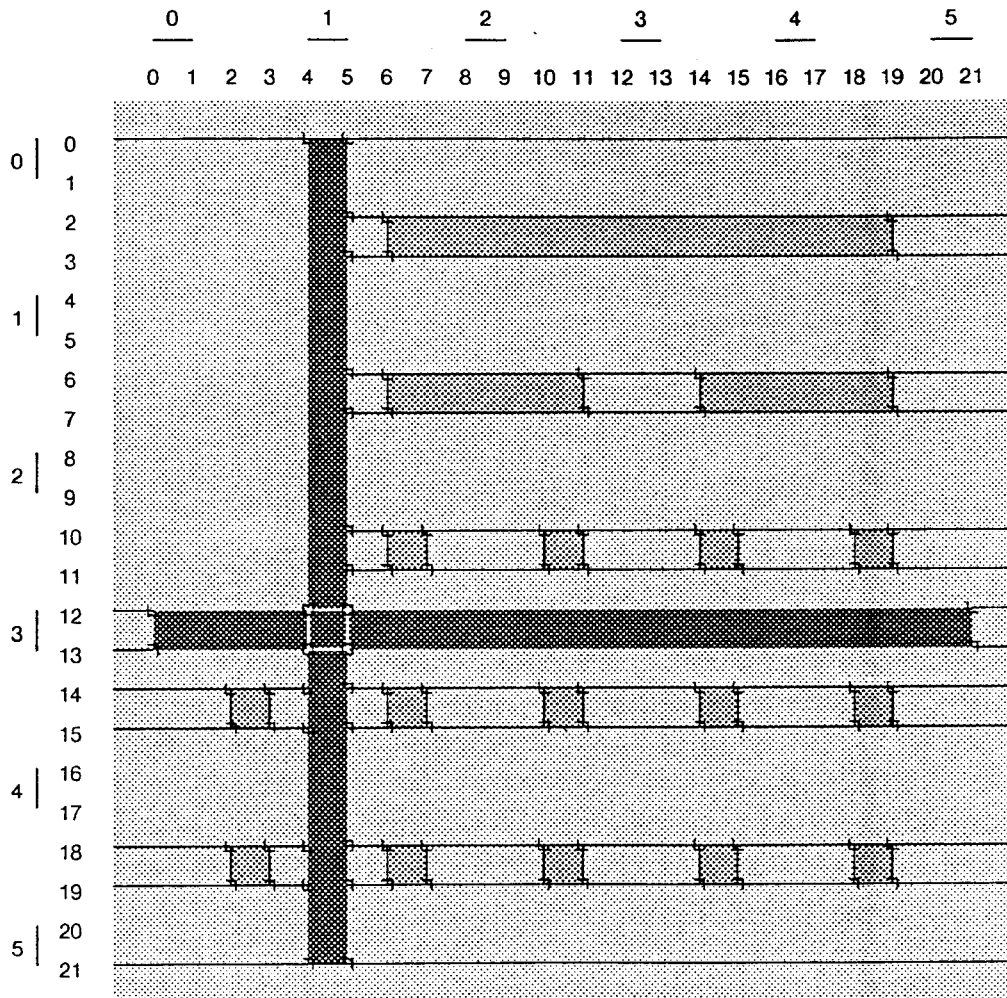


Figure 5-15. TABLE MAPPED ONTO A GRID DATA STRUCTURE for the table in Figure 5-6. The medium grey tiles correspond to table boxes and the dark grey tiles to typographic rules. Because of the way corner stitching coordinates are determined, grid modules are unit squares and grid boundaries are lines of unit width or height. The light grey shading represents the border tiles that hold the table entries together in the corner stitched data structure. The two rules intersect and are represented by several fragments, one of which is the intersection of the rules.

The external description of the table is parsed to create the internal data structure. The internal table description is held in a table object record, which remembers the table document subtree. Later, an updated external table description is spliced into the subtree after edits. The table object record also

points to the grid tessellation of table entries and there are fields for the grid positions and table origin that will be computed later by the table layout algorithm. The auxiliary constraints provided by the table designer are contained in a linked list.

```
TableRec: TYPE = RECORD[
    branchInDocument: REF DocumentObject, -- document subtree
    gridTessellation: REF Tessellation, -- area structure for tiles
    numberOfRows, numberOfColumns, INTEGER,
    rowGridPositions: ARRAY [0..numberOfRows) OF Dimension,
    columnGridPositions: ARRAY [0..numberOfColumns) OF Dimension,
    backgrounds: LIST OF REF TableEntryRec,
    origin: Position,
    constraints: LIST OF Constraint];
Dimension: TYPE = RECORD[value: REAL, units: Units];
Position: TYPE = RECORD[x,y: Dimension];
Units: TYPE = {in, cm, mm, pt, pc, ...};
DocumentObject: TYPE; -- representation of a document object
Tessellation: TYPE; -- representation of the grid data structure that contains TileRecs
Constraint: TYPE; -- representation of a linear inequality
```

The corner stitching data structure tessellates the plane with tiles. An initial tessellation contains a single tile that serves as a universe. A tile object contains four pointers for the major compass-points, a tessellation coordinate value, and a generic pointer to its associated data. (In Cedar, a generic pointer permits a data structure to be used in several different applications without the algorithm knowing the type of the data. The application is responsible for providing the pointer value and for later asserting the type of the pointer whenever the pointer is dereferenced to access the data fields.) New tiles are added by splitting the universal tile to maintain a "maximal East-West strip" property, which can be seen in Figures 5-14 and 5-15.

```
TileRec: TYPE = RECORD[
    north, south, east, west: REF TileRec,
    x, y: Coordinate, -- NE corner in the tessellation coordinate system
    data: REF ANY]; -- a REF TableEntryRec when used for table formatting
Coordinate: TYPE = INTEGER;
```

The table formatting prototype creates a table entry record for each corner stitched tile. The table entry contains the grid coordinates and the kind of table object. A box table entry points to the document object that will appear in the table, and retains the box dimensions, origin, and alignment point computed by



the layout algorithms. Typographic rules and background table entries have no content. The style attributes are remembered as a list of attribute-value pairs.

```

TableEntryRec: TYPE = RECORD[
  top, left: GridCoordinate, -- in the table grid coordinate system
  bottom, right: GridCoordinate,
  object: SELECT boxType: {box, rule, background} FROM
    box => [
      contents: REF DocumentObject,
      extents: BoxDimensions, -- computed later
      alignmentPoint: Position,
      origin: Position],
    rule => [],
    background => [],
  ENDCASE,
  styleAttributes: LIST OF Attributes
];
GridCoordinate: TYPE = INTEGER;
BoxDimensions: TYPE = RECORD[left, right, top, bottom: Dimension];
Attributes: TYPE; -- formatting attribute, value pair

```

### 5.4.3 Overlapping Planes

The corner stitching data structure is inherently planar, whereas some tables may involve overlapping table entries or rules. There are three things that may overlap: table entries that are “pasted over” other entries, rules that intersect with other rules, and backgrounds that underlay part or all of the table. The positions of all these overlapping elements can be expressed in terms of grid boundaries, so no additional layout information is necessary or generated by overlapped entries.

Backgrounds can be easily handled because there are generally few of them (often only one per table) and their size depends on and does not influence the table layout. It is sufficient to maintain a list of background rectangles to be shaded in the table object, so the rendering algorithm can traverse the list and paint the shaded background before rendering the rules and table entries.

Overlapping rules within tables are more interesting and more common. Rules intersect when a horizontal rule and a vertical rule meet. Even if the rules are of equal thicknesses, the end points of the rules must be adjusted to ensure that they avoid a notched corner, such as the leftmost examples in Figure 5-16. Additional complications arise when two rules of different colors or textures

intersect. Then one of the rules dominates and the other rule should have its endpoint adjusted to avoid penetrating the first, as shown by the middle examples in Figure 5-16. The prototype implementation only handles rules with square ends but an extension capability is available to add other intersection types.

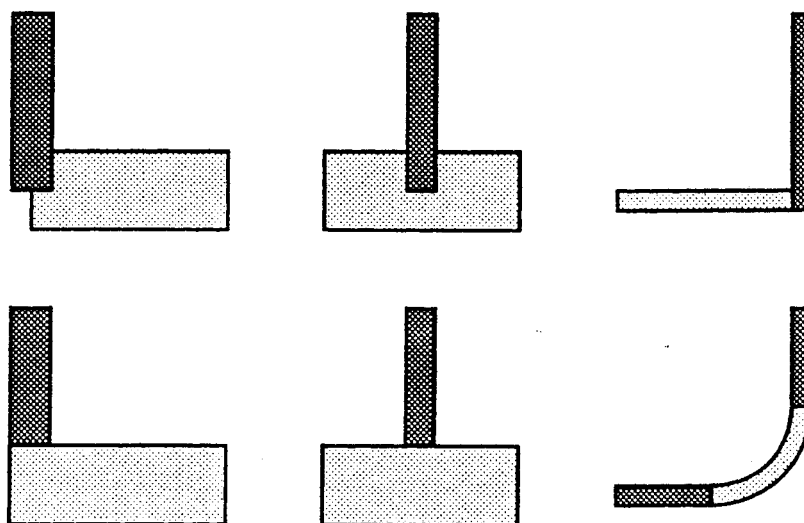


Figure 5-16. INTERSECTING RULES in a table may require one of several special treatments. Lines of different thicknesses; lines of different color; lines with repeating patterns (dashes, borders); lines with ornaments at the end; rounded corners where lines intersect with sufficient clearance are all examples of the treatments possible with this technique.

The fragmentation of corner stitched tiles, when a tile is about to penetrate an existing tile, provides the opportunity to capture the notion of intersecting rules explicitly. For example, the capability for rounded corners (as shown at the bottom right in Figure 5-16), or for placing special ornaments at the intersection of two rules can be specified at the intersection tile. The intersections between double or multiple rules can be treated in various ways. This technique for capturing the intersections between rules permits a general mechanism for specifying style attributes that apply to typographic rules and border patterns.

Overlapping table entries might be handled by maintaining an overlap order when they intersect. Table entry tiles in the grid data structure presently reference a single table entry. For overlapping entries, the tiles would have to maintain the list of overlapped table entries in overlap priority order. The rendering algorithm could then be modified to render the list of overlapped entries from back to front using the *Painter's algorithm*. These extensions for overlapping table entries have not been implemented in the prototype.

#### 5.4.4 Grid Algorithms

The main algorithms presented in this section are the table layout and table rendering algorithms. The table layout algorithm takes a table arrangement and produces the positions of the grid lines and table entries. The table rendering algorithm is responsible for creating a printable or displayable version of the table given the positions and content of the table. These two algorithms require an enumeration algorithm to act on all of the table entries in the internal table data structure. Interactive manipulation of tables requires a hit-testing algorithm, which resolves the table entry that is being pointed at on the screen, and dynamic restructuring algorithms to insert and delete parts of a table.

##### *Table Layout Algorithm*

The table layout algorithm determines the coordinates (relative to a table origin) of the grid lines given the grid data structure representing the table arrangement. The layout algorithm also needs the bounding box dimensions and alignment point of each table entry, which are obtained through the object layout procedure associated with the class of document content in that table entry. Typographic rules also have widths which are obtained through the style machinery. Linear inequality constraints describe the positioning relationships between table entries and grid lines. A mathematical constraint solver is described later in Section 5.5, where the complete table layout algorithm is presented. The time complexity of the layout algorithm is dominated by the algorithm for satisfying the linear inequality constraints.

##### *Table Rendering Algorithm*

The table must be rendered in a form suitable for display or printing. The table rendering algorithm takes the grid positions determined by the table layout algorithm along with the collection of table entries in the grid data structure and instantiates each table entry at its appropriate position. The content of each table entry is rendered via the rendering procedure associated with that document object class.

To handle the possible overlap of data, the table content is rendered in back to front order: shaded backgrounds first, then rules, then table entries.

The table rendering algorithm uses a device independent imaging model [Warnock&Wyatt, CedarGraphics] for raster devices to produce formatted output for both displays and printing devices. Device independence provides a WYSIWYG capability for interactive editing of tables because the display screen presents the

same fonts and graphics as the printed form of the table, subject to differences in device resolution. The implementation of a table editor shares the table rendering algorithm with the document formatter.

*Algorithm R (Table Rendering)*

- R1 [Output Backgrounds] Traverse the list of shaded backgrounds in the table and for each one, output a shaded rectangle with corners determined by the coordinates of the table entry's four grid line positions, and with the color specified by the area color style attribute.
- R2 [Enumerate Rule Tiles] Traverse the table grid structure and render each table rule:
  - R2.1 [Draw Rule] Determine the rule shape from the grid line position, the rule thickness specified by a style attribute, and any intersections with other rules. Draw the rule in the color specified by the rule color style attribute.
- R3 [Enumerate Table Entries] Traverse the table grid structure and render each table entry:
  - R3.1 [Output Table Content] Invoke the rendering procedure for this table object at its origin  $(x,y)$ .

One might extend the rendering algorithm for interactive use so that it performs only minimal repainting. The technique is similar to that used by WYSIWYG text editors. By keeping state information about table entries that change, only those entries need be repainted. By keeping track of grid positions that change, a block-move instruction on portions of the bit-mapped screen permits opening up (or closing up) grid lines to update the table without having to rerender the entire table. The table editor would have to cooperate with the table rendering algorithm to maintain these state variables.

*Enumerate Area Algorithm*

Both the table layout and the table rendering algorithm must enumerate all of the table entries and rules in the grid data structure. The corner stitching data structure has a directed area enumeration algorithm [Ousterhout, Corner Stitching] that takes linear time in the number of tiles enumerated. The algorithm is given the corner stitching data structure and a rectangular search area.

*Algorithm EA (Enumerate Area Algorithm)*

- EA1 [Locate Lower Left Tile] Use the corner-stitching point-finding algorithm to locate the tile at the lower left corner of the search area.
- EA2 [Walk Left Edge] Walk the tiles along the left edge and for each tile encountered invoke the following steps recursively:
  - EA2.1 [Enumerate This Tile] Perform some user-supplied action on this tile, or if no action was supplied, then append this tile to an enumeration list.
  - EA2.2 [Check Right Boundary] If the right edge of this tile is outside the search area, then return.
  - EA2.3 [Locate Right Neighbors] Use the corner-stitching neighbor-finding algorithm to enumerate neighbors along the right edge of this tile. For each neighbor that intersects the search area:
    - EA2.3.1 [Recurse on Neighbor] If the top-left corner of the neighbor touches the current tile, or if the top edge of the search area cuts both the current tile and its neighbor, then recurse at step EA2.1 on the neighbor.

The implementation of the enumeration algorithm used in the table formatting prototype uses a nonrecursive algorithm due to Shand [Shand, CornerStitching] that is based on traversing the tiles like a threaded tree. The implementation requires the caller to supply an action procedure to be performed for each tile. The enumeration algorithm is used in the table layout, table rendering, deletion, and insertion algorithms. For example, the table rendering algorithm supplies an action procedure that renders the content of a particular table entry.

*Hit-testing Algorithm*

A hit-testing algorithm is necessary for interactive manipulation of a table. The hit-testing algorithm determines the table entry corresponding to a position on the display screen pointed at by the user. The algorithm must be quite rapid if the table entry is to be continuously highlighted as the user moves the pointing device across the table on the screen. The algorithm depends on a corner stitching algorithm for locating a corner stitched tile given the tile coordinates. The corner stitching algorithm takes  $O(\sqrt{n})$  time on average when

tiles have a relatively uniform size and  $O(n)$  time in the worst case when all the tiles are in a single column or row [Ousterhout, Corner Stitching].

Given the screen  $(x,y)$  coordinates of the pointing device, the hit-testing algorithm first converts the screen coordinates relative to the display window into table coordinates relative to the table origin. The appropriate row and column grids are found by searching the table grid position arrays. The grid coordinates are converted into corner stitched coordinates and the tile locating algorithm returns the table entry tile that contains the data pointer to the table entry object.

*Algorithm H (Hit-testing Algorithm)*

- H1 [Convert Coordinates] Given  $(S_x, S_y)$ , the screen coordinates, subtract the screen coordinates of the table origin,  $(O_x, O_y)$ , relative to the display window, to produce the table coordinates  $(T_x, T_y)$  of the point. Report an error if  $(T_x, T_y)$  lies outside the table.
- H2 [Find row and column grids] Search the two vectors of grid coordinates for row grid line  $G_r$  preceding  $T_x$  and the column grid line  $G_c$  preceding  $T_y$ . This can be done with a logarithmic search.
- H3 [Find Table Entry] Locate the grid tile at the grid coordinates  $(G_r, G_c)$  using the corner stitching location algorithm and extract the reference pointer to the resulting table entry.

*Delete Algorithm*

Two algorithms for dynamically inserting and deleting table elements were incorporated into the table formatting prototype. The development of interactive tools for table editing was not central to the research in this thesis, but the tools were useful for constructing examples and gaining insight into the value of the grid structure for interactive editing. Chapter 6 contains plans to implement better algorithms based on better data structures.

There are two variations of the deletion algorithm. The first variant deletes table entries without changing the table topology, and the second deletes grid lines and hence changes the topology.

In the first variant that deletes only table entries, the deletion algorithm is given two pairs of grid lines that determine a rectangular region. By enumerating the area within that rectangular region all of the table entries intersecting that region are found. Some entries may be partially outside the region and therefore should not be deleted. This algorithm takes time linear in

the number of entries enumerated in the rectangular region due to the behavior of the area enumeration algorithm.

*Algorithm E (Table Entry Deletion Algorithm)*

- E1 [Enumerate Area to Delete] Enumerate all the table entries within the two pairs of grid lines, and for each table entry:
  - E1.1 [Delete Table Entry] If the table entry is completely within the two pairs of grid lines, then delete the table entry from the grid data structure.

The second variant of the deletion algorithm deletes grid lines and all table entries between them. The algorithm is given a pair of grid lines. All grid lines between the pair of grid lines (inclusive) are deleted and replaced by a single grid line. All table entries found entirely within the pair of grid lines are deleted. Any table entry that crosses outside the pair of grid lines is changed to reflect the new topology. This deletion algorithm is presented in its simplest form, which creates a new copy of the grid data structure.

*Algorithm G (Table Grids Deletion Algorithm)*

- G1 [Prepare New Grid Structure] Prepare a second grid data structure to receive the modified table topology.
- G2 [Enumerate the Old Grid] Use the area enumeration algorithm to find all the table entries that intersect the region defined by the pair of grid lines. For each table entry, perform one of the following actions:
  - G2.1 [Table Entry Within Grid Pair] Ignore this table entry; it will not appear in the new structure and is not copied.
  - G2.2 [Table Entry Precedes Grid Pair] Create a tile in the new grid structure and copy this table entry.
  - G2.3 [Table Entry Straddles Grid Pair] Adjust the farthest grid coordinate of this table entry by the number of grid lines deleted, and create a tile in the new grid structure for this changed table entry.
  - G2.4 [Table Entry Follows Grid Pair] Adjust both grid coordinates of this table entry by the number of grid lines deleted, and create a tile in the new grid structure for this changed table entry.
- G3 [Install New Grid Structure] Update the table object to use this new grid structure.

The implementation of the deletion algorithm in the table prototype relies on the existing corner stitching implementation and avoids designing or implementing a more dynamic data structure suggested in Chapter 6. The deletion algorithm uses two copies of the table grid structure and ping pongs between them. The time to create the new grid structure is dominated by the time needed to refresh the display screen. A major performance benefit is avoiding automatic storage collection, since the corner stitching implementation caches deleted grid tiles and reuses them without allocating new ones.

### *Insert Algorithm*

The insertion algorithm also comes in two variants, (a) one which inserts a table entry into an empty grid module without changing the topology, and (b) another which inserts a new grid line that changes the table topology. In the insert table entry variant (a), the new table entry is created and inserted into the grid structure at the grid coordinates corresponding to an empty tile. In the insert grid line variant (b), the dual grid structure technique shown above is used to create a new table topology.

## 5.5 Table Layout via Constraints

A linear constraint solver is employed to determine the table geometry from the table topology, the content of table entries, and explicit auxiliary constraints supplied by the table creator. The grid data structure represents the topology of all the table entries and contains pointers to their content. The layout algorithm enumerates all of the table entries and generates appropriate constraints on their positions. Additional constraints may be supplied with the table to impose special conditions such as forcing columns of equal width. Solving the resulting system of linear inequalities produces the computed positions of all the grid lines and table entries.

### 5.5.1 Constraint Systems

Examples of constraint systems in computer graphics provided the motivation for this technique. Sutherland's *Sketchpad* [Sutherland, Sketchpad] was a seminal interactive graphics system that relied on constraint satisfaction to position graphical elements. Borning's implementation of *Thinglab* [Borning, Thinglab] demonstrated the use of constraints in several problem domains. The document layout and document content examples in his report were of particular



interest. The layout example [Borning, Thinglab, p29] showed a paragraph surrounded by a rectangle precisely large enough to hold the paragraph. When the width of the rectangle or content of the paragraph was changed, the rectangle's height was adjusted accordingly.

Several other graphics systems used constraints to position things. JUNO [Nelson, Juno] and *ideal* [van Wyk, ideal] both use nonlinear constraints to determine the position of graphical objects. *pic* [Kernighan, pic] labels the corners of boxes and position objects relative to those labellings. The graphical debugger Incense [Myers, Incense] contained a heuristic for positioning nested objects in a data structure by dividing the available space in two. The ability of these systems to position parts of an illustration relative to one another was very attractive and analogous to the layout of table entries relative to one another.

Linear inequalities are sufficient to describe the positioning relationships between table entries, such as making a column wide enough for an entry, or making a row heading tall enough to span several rows. Restricting the constraints to linear inequalities prevents expressing area relationships, but it is more common to express such relationships by establishing a ratio among the sides of the rectangle which can be described by linear inequalities. All the distances in a table can be expressed as rectilinear (horizontal or vertical) distances. Thus addition, subtraction and scalar multiplication are the only operations needed to express the table positioning relationships.

Using inequalities introduces the notion of slack variables that turn the inequality into an equality for the constraint solver. These slack variables capture the extra room left over for the table entry to fit the grid lines. Smaller table entries have greater slack than larger entries, and one or more table entries will have zero slack. The slack variables are helpful in determining whether the constraint system must be recomputed when editing a table. A table entry with slack may grow or shrink without requiring the constraint system to be recomputed, while a table entry with zero slack that changes size will always require recomputing the grid lines.

The linear inequality solver due to Nelson [Nelson, Program Verification] used in the table formatting prototype has an useful property for implementing an undo facility in an interactive table editor. The constraint tableau maintains a stack of 'dead' tableau columns which permits a previous state of the tableau to be recomputed quickly. Changes to the table that modify table layout constraints may be quickly undone through this mechanism without recomputing the entire solution of the constraint system.

### 5.5.2 The Complexity of Linear Constraint Solvers

Solving a system of linear inequalities is equivalent to the LINEAR PROGRAMMING problem, which is known to be solvable in polynomial time [Khachiyan, LP]. A table arrangement without centered table entries can be described by a system of linear inequalities using only two variables per inequality, as described in the next section. This particular LINEAR INEQUALITY problem for two variables can be solved in  $O(n^3)$  worst case time [Aspvall&Shiloach, Linear Inequalities], where  $n$  is the number of constraints. Algorithms for solving more general systems of linear inequalities based on the Simplex method have  $O(n^3)$  expected time behavior [Dantzig, LP]. A version of the Simplex algorithm due to Nelson [Nelson, Program Verification] was implemented in the table formatting prototype.

### 5.5.3 The Constraint Table Layout Problem

The table layout algorithm introduced in Section 5.4.4 depends on a linear inequality constraint solver to compute the positions of the grid and table entries. This section discusses the linear inequalities actually used to describe the table arrangement.

Consider a particular table entry represented as a box  $k$ . This table entry is positioned within a pair of horizontal grid lines and a pair of vertical grid lines as shown in Figure 5-17. The alignment point of the box is to be aligned with other boxes that share the same pair of grid lines. Label the four grid lines surrounding the box as  $leftGrid_k$ ,  $rightGrid_k$ ,  $topGrid_k$ ,  $bottomGrid_k$ . Label the offsets from the alignment point to the edge of the box as distances  $leftOffset_k$ ,  $rightOffset_k$ ,  $topOffset_k$  and  $bottomOffset_k$  (these offsets are fixed by the content and can be determined before table layout). The position of the alignment point within box  $k$  will be  $(X_k, Y_k)$ .

The vertical grid lines represent column boundaries and their positions are designated by the variables  $C_i$ . The horizontal grid lines represent row boundaries and their positions are designated by the variables  $R_j$ . Boxes within the pair of grid lines  $left$  and  $right$  that are vertically aligned will be positioned at the variable  $V_{left,right}$ , while boxes horizontally aligned within the row between grid lines  $top$  and  $bottom$  will be positioned at  $H_{top,bottom}$ . All variables are restricted to positive values.

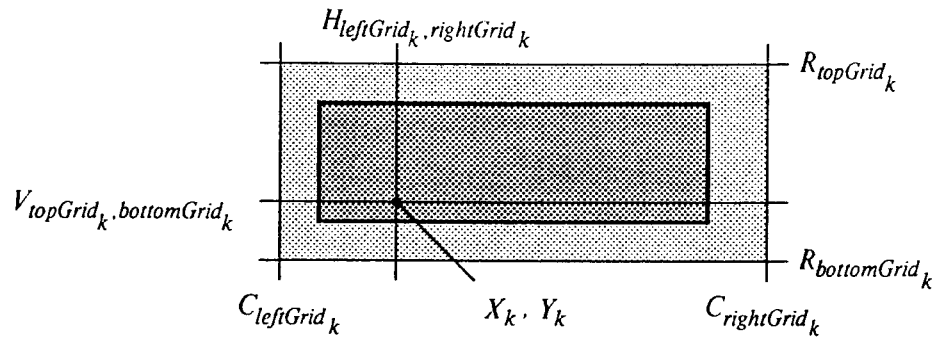


Figure 5-17. CONSTRAINT VARIABLES FOR A PARTICULAR TABLE ENTRY are shown in this diagram. The box representing the table entry is in dark grey and the grid module formed by two pairs of grid lines is shown in light grey, the column lines  $C_{left}$  and  $C_{right}$  and the row lines  $R_{top}$  and  $R_{bottom}$ . The alignment point of the table entry box goes through the two alignment lines,  $V_{left,right}$  and  $H_{top,bottom}$ .

The objective function for the constraint solver will be to minimize the width and depth of the table. For a table with  $m$  rows and  $n$  columns, the objective function would be

$$\text{Min } \{(C_n - C_0) + (R_m - R_0)\} .$$

The table layout the relationships among grid lines and alignment points can be described by inequality constraints. The first set of constraints (1) ensure that all of the grid lines are in the expected order and that the alignment line is within the column. One set of constraints is required for each pair of grid lines, *left* and *right*, that contain a table entry. (Column constraints are used in the examples that follow; row constraints are similar.)

$$\begin{aligned} C_{right} - C_{left} &\geq 0 \\ V_{left,right} - C_{left} &\geq 0 \\ C_{right} - V_{left,right} &\geq 0 . \end{aligned} \quad (1)$$

A constraint to ensure that the grid lines are sufficiently far apart to contain the width of a table entry is generated for each table box  $k$ :

$$C_{rightGrid_k} - C_{leftGrid_k} \geq leftOffset_k + rightOffset_k . \quad (2)$$

In practice, these two sets of constraints are not generated explicitly since they are subsumed by the alignment constraints below.

The alignment constraints vary with the type of alignment requested. Aligning a table box flush left in the column without regard to any other box in

the column generates these two constraints to position the box immediately adjacent to the column grid:

$$\begin{aligned} X_k - C_{leftGrid_k} &= leftOffset_k & (3L) \\ C_{rightGrid_k} - X_k &\geq rightOffset_k . \end{aligned}$$

Similarly, aligning a table box flush right in the column without regard to any other box switches the equality and inequality constraints from those in (3L):

$$\begin{aligned} X_k - C_{leftGrid_k} &\geq leftOffset_k & (3R) \\ C_{rightGrid_k} - X_k &= rightOffset_k . \end{aligned}$$

Centering a table box within the column may involve one of two centering concepts. The first forces the box alignment point to be equidistant between the column grid lines:

$$2 \cdot X_k - C_{leftGrid_k} - C_{rightGrid_k} = 0 . \quad (3C)$$

The second centering concept positions the box to balance the excess whitespace surrounding the box. The excess whitespace is the width of the column ( $C_{rightGrid_k} - C_{leftGrid_k}$ ) less the width of the box ( $leftOffset_k + rightOffset_k$ ):

$$\begin{aligned} X_k - leftOffset_k - C_{leftGrid_k} &= \\ &1/2 \cdot [(C_{rightGrid_k} - C_{leftGrid_k}) - (leftOffset_k + rightOffset_k)] \end{aligned}$$

which can be simplified to the form:

$$2 \cdot X_k - C_{leftGrid_k} - C_{rightGrid_k} = leftOffset_k - rightOffset_k . \quad (3C')$$

The two centering constraints (3C) and (3C') are equivalent only if the two offsets are equal, which is when the alignment point is in the center of the box.

Note that up until now none of these alignment constraints involved aligning a set of boxes with each other, only positioning the box within the column. To align several boxes, we need to force the alignment point within a box to be equal to the position of the alignment line within that column. Thus, for each box  $k$  in the set, generate an equality constraint:

$$X_k - V_{leftGrid_k, rightGrid_k} = 0 \quad (4)$$

In fact, if the variable  $X_k$  does not appear in any other constraint, then it may be dispensed with and treated as an alias.

The positioning constraints for each box aligned within a set are simpler because they only require that the box fit within the column:

$$\begin{aligned} V_{leftGrid_k, rightGrid_k} - C_{leftGrid_k} &\geq leftOffset_k \\ C_{rightGrid_k} - V_{leftGrid_k, rightGrid_k} &\geq rightOffset_k . \end{aligned} \quad (5)$$

However, these constraints are not sufficient to force the set of boxes to be flush left, flush right, or centered within the column. This must be accomplished by guiding the constraint solver to a particular desired solution. For example, to force the set of aligned boxes flush left, the objective function to be minimized by the constraint solver must be augmented with a term for the left distance:

$$\text{Min } \{(V_{leftGrid_k, rightGrid_k} - C_{leftGrid_k})\} . \quad (6)$$

A similar term is necessary to force the set of aligned boxes flush right.

Centering the alignment line for a set of boxes within a column may be accomplished in two ways. The first simply positions the alignment line equidistant between the column grid lines and does not involve an additional term in the objective function:

$$2 \cdot V_{leftGrid_k, rightGrid_k} - C_{leftGrid_k} - C_{rightGrid_k} = 0 . \quad (7)$$

The second centering technique is to balance the whitespace between the set of aligned boxes and the grid lines. The objective function requires a new variable equal to the maximum whitespace in the column. This was not done in the prototype but is considered later in section 5.5.4 when such 'maximum' variables are introduced to handle large tables.

These linear constraints can be generated automatically for a table from the information kept in the grid data structure. A small expression language for constraint inequalities is provided in the prototype for the table designer to describe additional constraints for special effects. For instance, to create two columns of equal-width, say the column between grid lines *left* and *right* and the column between *left'* and *right'*, the following equality constraint is added to the table specification:

$$(C_{left'} - C_{right'}) - (C_{left} - C_{right}) = 0 . \quad (8)$$

The expression language accepts a standard form of the variable names. Two such constraints were included in the external representation shown in Figure 5-13. A variety of user interfaces to this constraint expression facility are possible. Only the simplest textual interface of typing the constraint equations is supported in the prototype.

---

$H_{4,5} - R_4 \geq 6.225697$	(flush left box 4,0)
$R_4 - H_{4,5} \geq 15.25376$	
$H_{3,4} - R_4 \geq 6.225697$	(flush left box 3,0)
$R_3 - H_{3,4} \geq 14.97279$	
$H_{4,5} - R_5 \geq 6.0$	(flush left box 4,4)
$R_4 - H_{4,5} \geq 12.22937$	
$H_{4,5} - R_5 \geq 6.0$	(flush left box 4,3)
$R_4 - H_{4,5} \geq 12.22937$	
$H_{4,5} - R_5 \geq 6.0$	(flush left box 4,2)
$R_4 - H_{4,5} \geq 12.22937$	
$H_{4,5} - R_5 \geq 6.0$	(flush left box 4,1)
$R_4 - H_{4,5} \geq 12.22937$	
$H_{3,4} - R_4 \geq 6.0$	(flush left box 3,4)
$R_3 - H_{3,4} \geq 12.22937$	
$H_{3,4} - R_4 \geq 6.0$	(flush left box 3,3)
$R_3 - H_{3,4} \geq 12.22937$	
$H_{3,4} - R_4 \geq 6.0$	(flush left box 3,2)
$R_3 - H_{3,4} \geq 12.22937$	
$H_{3,4} - R_4 \geq 6.0$	(flush left box 3,1)
$R_3 - H_{3,4} \geq 12.22937$	
$H_{2,3} - R_3 \geq 6.0$	(flush left box 2,4)
$R_2 - H_{2,3} \geq 15.25376$	
$H_{2,3} - R_3 \geq 8.257013$	(flush left box 2,3)
$R_2 - H_{2,3} \geq 15.25376$	
$H_{2,3} - R_3 \geq 6.0$	(flush left box 2,2)
$R_2 - H_{2,3} \geq 15.25376$	
$H_{2,3} - R_3 \geq 8.257013$	(flush left box 2,1)
$R_2 - H_{2,3} \geq 15.25376$	
$H_{1,2} - R_2 \geq 8.482721$	(flush left box 1,3)
$R_1 - H_{1,2} \geq 15.25376$	
$H_{1,2} - R_2 \geq 8.482721$	(flush left box 1,1)
$R_1 - H_{1,2} \geq 15.25376$	
$H_{0,1} - R_1 \geq 8.482721$	(flush left box 0,1)
$R_0 - H_{0,1} \geq 15.25376$	

---

Figure 5-18. THE ROW CONSTRAINTS GENERATED for the table in Figure 5-6. The variables for the row grid line positions are  $R_0, R_1, R_2, R_3, R_4, R_5$  and the horizontal alignment line positions are  $H_{0,1}, H_{1,2}, H_{2,3}, H_{3,4}, H_{4,5}$ .

---

---

$2.0 \cdot C_1 - C_0 - C_3 = 0$	
$2.0 \cdot C_2 - C_1 - C_3 = 0$	
$C_0 - V_{0,1} \geq 3.0$	(flush left box 4,0)
$C_1 - V_{0,1} \geq 69.31976$	
$C_0 - V_{0,1} \geq 3.0$	(flush left box 3,0)
$C_1 - V_{0,1} \geq 55.72097$	
$C_5 - C_4 \geq 24.0675$	(center box 4,4)
$2.0 \cdot V_{4,5} - C_4 - C_5 = 0$	
$C_4 - C_3 \geq 60.2025$	(center box 4,3)
$2.0 \cdot V_{3,4} - C_3 - C_4 = 0$	
$C_3 - C_2 \geq 24.0675$	(center box 4,2)
$2.0 \cdot V_{2,3} - C_2 - C_3 = 0$	
$C_2 - C_1 \geq 60.2025$	(center box 4,1)
$2.0 \cdot V_{1,2} - C_1 - C_2 = 0$	
$C_5 - C_4 \geq 24.0675$	(center box 3,4)
$2.0 \cdot V_{4,5} - C_4 - C_5 = 0$	
$C_4 - C_3 \geq 60.2025$	(center box 3,3)
$2.0 \cdot V_{3,4} - C_3 - C_4 = 0$	
$C_3 - C_2 \geq 24.0675$	(center box 3,2)
$2.0 \cdot V_{2,3} - C_2 - C_3 = 0$	
$C_2 - C_1 \geq 60.2025$	(center box 3,1)
$2.0 \cdot V_{1,2} - C_1 - C_2 = 0$	
$C_5 - C_4 \geq 30.86088$	(center box 2,4)
$2.0 \cdot V_{4,5} - C_4 - C_5 = 0$	
$C_4 - C_3 \geq 63.23784$	(center box 2,3)
$2.0 \cdot V_{3,4} - C_3 - C_4 = 0$	
$C_3 - C_2 \geq 30.86088$	(center box 2,2)
$2.0 \cdot V_{2,3} - C_2 - C_3 = 0$	
$C_2 - C_1 \geq 63.23784$	(center box 2,1)
$2.0 \cdot V_{1,2} - C_1 - C_2 = 0$	
$C_5 - C_3 \geq 88.15895$	(center box 1,3)
$2.0 \cdot V_{3,5} - C_3 - C_5 = 0$	
$C_3 - C_1 \geq 73.83745$	(center box 1,1)
$2.0 \cdot V_{1,3} - C_1 - C_3 = 0$	
$C_5 - C_1 \geq 159.6822$	(center box 0,1)
$2.0 \cdot V_{1,5} - C_1 - C_5 = 0$	

---

Figure 5-18 (continued). THE COLUMN CONSTRAINTS GENERATED for the table in Figure 5-6. The first two constraints are the auxiliary constraints to force the equal column widths; the rest are generated by the table layout algorithm. The variables for the column grid line positions are  $C_0, C_1, C_2, C_3, C_4, C_5$  and the vertical alignment line positions are  $V_{0,1}, V_{1,2}, V_{1,3}, V_{1,5}, V_{2,3}, V_{3,4}, V_{3,5}, V_{4,5}$ .

---

### 5.5.4 Handling Large Tables

The constraint scheme just outlined suffers from rapid growth in the problem size as the tables grow larger. Several strategies can greatly reduce the number of constraints generated.

The first strategy eliminates redundant variables whenever they appear as aliases of other variables. For example, the box position  $X_k$  is often an alias for  $V_{leftGrid_k, rightGrid_k}$  when they are set equal and therefore  $X_k$  can be eliminated.

In many table layouts, a simplifying assumption may be made to solve the row and column constraints separately. Treating the two sets of constraints independently reduces the number of constraints and the size of the corresponding constraint tableau. It is worth assessing the necessity of this assumption and the situations in which it is violated.

The independence assumption does not hold when one wants to create grid designs with modules that are square or have specific aspect ratios. A square module results from constraining the row height to equal the column width. Obviously, in this situation the sets of row and column constraints are not independent. The assumption is also not true when trying to distribute whitespace within a table, for example, by folding wide horizontally-oriented table entries to trade off more vertical depth for less horizontal width. In this situation, decreasing the column width increases the row height and the constraints are again not independent.

The independence assumption can be tested automatically by examining the variables in each constraint. Constraints that mix row and column variables, such as one specifying that a row height equals a column width, are interdependent and must be solved simultaneously. Constraints that do not mix variables are independent and can be solved separately as smaller problems. The prototype currently assumes that the row and column constraints are independent, but one could easily check the explicitly supplied constraints for violations of this assumption and thus determine automatically whether to solve the table layout constraints independently or not.

What is the cost of solving interdependent row and column constraints? When the two sets are combined into a single constraint system the problem size is doubled. Since the constraint solver takes  $O(n^3)$  time on average, the combined constraint layout would require about four times as much as the sum of the two smaller independent computations.



A reduction in the number of constraints can also be achieved by observing the structure of the constraint inequalities. The constraints that determine the grid positions surrounding a particular table entry are all similar:

$$\begin{aligned} V_{leftGrid_k, rightGrid_k} - C_{leftGrid_k} &\geq leftOffset_k \\ C_{rightGrid_k} - V_{leftGrid_k, rightGrid_k} &\geq rightOffset_k . \end{aligned} \quad (5)$$

For the set of table entries within a particular column, these inequalities are dominated by a single maximum value of the right hand side. The maximum value could be determined by simpler means than expressing so many redundant constraints. We can discover the value by a linear preprocessing pass through all the table entries in that particular column to determine the maximum left and right offsets. The two maximum values are assigned to new variables used by the constraint solver:

$$\begin{aligned} maxLeft_{left, right} &= \max\{leftOffset_k \text{ for all boxes } k \text{ in column } left, right\} \\ maxRight_{left, right} &= \max\{rightOffset_k \text{ for all boxes } k \text{ in column } left, right\} . \end{aligned}$$

With these maximum values known by these variable names, the set of inequalities per column (one or more constraint for each table entry) can be replaced by one inequality per column. This vastly reduces the number of constraints from the product of the number of entries per column times the number of columns to simply the number of columns:

$$\begin{aligned} V_{left, right} - C_{left} &\geq maxLeft_{left, right} \\ C_{right} - V_{left, right} &\geq maxRight_{left, right} . \end{aligned} \quad (5')$$

A further optimization in the linear constraint solver can be made based on the observation that most of the constraint inequalities have only two variable terms. Thus, sparse matrix techniques might reduce the space requirements of the solver at the expense of more overhead in the constraint solver. The prototype implementation uses a conventional matrix structure for the constraint tableau.

### 5.5.5 The Layout Algorithm

The complete table layout algorithm including the constraint solver can now be presented. The algorithm requires the grid data structure for the table arrangement and the contents of all of its table entries. The first step in the algorithm ensures that the dimensions of all of the table entries are known. These dimensions are provided by invoking the particular layout algorithm for each table entry object. In the event that a table entry contains another

subtable, then the layout algorithm would be invoked on that subtable first. The dimension information for each table entry is cached in the table entry object since the box dimensions are used twice when the row and column constraints are independent.

*Algorithm L (Table Layout)*

- L1 [Dimension the Boxes] Determine the dimensions of the table entries by invoking the layout procedure for that class of object. Remember the dimensions since they are used for both row and column layout.
- L2 [Layout the Rows]
  - L2.1 [Establish Row Constraints] Enumerate the table grid structure and generate the appropriate constraints for each table entry.
  - L2.2 [Solve] Invoke the constraint solver on the row constraints, and record the row grid coordinates.
  - L2.3 [Position Table Entries] Enumerate the table grid structure and compute the layout coordinates of the origin for each table entry from the computed grid positions.
- L3 [Layout the Columns]
  - L3.1 [Establish Column Constraints] Enumerate the table grid structure and generate the appropriate constraints for each table entry.
  - L3.2 [Solve] Invoke the constraint solver on the column constraints, and record the column grid coordinates.
  - L3.3 [Position Table Entries] Enumerate the table grid structure and compute the layout coordinates of the alignment position for the each table entry from the computed grid positions.

## 5.6 Conclusions

This chapter has introduced a new framework for interactive table formatting. The separation of table topology from table geometry has been incorporated into this framework. A grid structure describes the topological table arrangement and a mathematical constraint solver computes the table layout. A prototype table formatter was built using extensions to the Tioga

document model and to the style machinery. Unlike many other document formatting systems, this approach allows more general table designs to be described and edited with interactive tools.

The grid structure provides an explicit representation of the table topology suitable for WYSIWYG interactive editing of tables. The grid structure represents areas directly, permitting symmetric treatment of rows and columns, and enabling typographic rules and background areas to be treated explicitly. An implementation of the grid structure exists with linear algorithms suitable for interactive manipulation which support selection hierarchies of table structures, table entries, row, columns, and the entire table. Support for overlapping table designs is possible; backgrounds are easy.

The mathematical constraint solver computes the table geometry from linear inequalities that express the table alignment possibilities. Most of the inequalities are generated automatically from alignment and positioning style attributes, while additional constraints may be specified by the table designer to enforce special layout requirements.

The table object class in the document model permits tables to exist within documents and be interactively edited by appropriate class editors. Tables may contain arbitrary content since a table entry is treated as an arbitrary document object. Style attributes for tables are managed by extensions to the style machinery. New attributes for table alignment and various typographic features unique to tables have been added to the style mechanism of Chapter 3. A search path appropriate to the row/column structure of tabular information determines the values of attributes in force at each table entry.

The prototype table formatter worked well for the tables in this thesis. All of the tables in Chapter 4, the complex table in Figure 5-3, and the remaining tables in Chapter 5 were formatted by the prototype. The grid system proved to be a simple concept to work with, and creating additional linear inequality constraints for special effects appears to have considerable promise for extending the kinds of achievable table designs. Several of the large tables became tedious to manipulate in their textual form, a complaint that will be remedied with the completion of an interactive table editor. An implementation of an interactive table editor within Tioga based on this prototype is underway. Goals of this further work are described in Chapter 6.

# 6

## Future Directions

---

This thesis has presented two prototypes that were implemented for composing complex document content, one prototype for graphical style and the other for table formatting with grids and constraints. These experiments led to many implementation issues and several new research problems. The implementation issues arose from a desire to provide integrated and interactive illustration and tabular composition tools in the Tioga and Cedar environments at Xerox PARC. The research problems follow from a desire to unify the two approaches into the same document model and to extend the range of functionality in both approaches. This chapter outlines some of these future problems and issues, in particular a comprehensive document framework, interactive user interfaces for the prototypes, various problems with graphical style and table formatting, and a proposal for considering rule-based expert system support for problems in document composition.

### 6.1 Document Model

The document model has been a common thread through the two prototypes for graphical style and table formatting. This experience should be generalized into a document model that can integrate arbitrary classes of document content by more rigorously defining the object interface for the editor and the formatter. An interactive editor for such a document model would coordinate various special purpose editors for each class of document content

and would provide shared undo/redo facilities. Several difficulties arise with designing a coordinated user model of interaction, specifying the notion of selection in nontextual objects, and arbitrating interaction requests between overlapping objects. A more general document structure than a hierarchical one similar to the abstract document object model from Kimura [Kimura, thesis] should be considered to incorporate objects with arbitrary interlinking, such as footnotes, floating figures or tables with several references, and running headers.

## 6.2 Graphical Style

The graphical style prototype would benefit from an interactive design environment for specifying styles. The notion of interactive property sheets from the Xerox Star can be extended to support styles by naming groups of properties and by managing dictionaries of style rules. Additional graphic design tools for layouts and rendering algorithm specifications are necessary. Other user interface issues are discussed below.

The suite of illustration rendering algorithms in the graphical style prototype provides a basis for an extensible mechanism for adding new algorithms. Initial uses of this mechanism might be to add new algorithms for border patterns, shadows on text objects, or special color highlights. Revealing different levels of detail as a style attribute when changing media could be requested as an additional view specification, similar to level clipping when presenting hierarchical documents.

The ideas of graphical style should be incorporated into a new illustration system, perhaps based on the Griffin user interface and editing model. This new illustrator could incorporate layout constraints by utilizing constraints as demonstrated in the Juno [Nelson, Juno] and Gob [Zabula-Saelles, GOB] illustration systems and as incorporated in the table formatting prototype described in this thesis.

## 6.3 Table Formatter Implementation

The table formatting prototype needs several implementation improvements. The corner stitching data structure implementation should be replaced by a dynamic topological data structure so that the insert and delete editing operations are implemented without copying. The minimal repainting algorithm

for incremental interactive updates to the The handling of constraints for tables should gracefully solve two sets of row and column constraints when the constraints do not interact, but should solve the combined constraint problem when they do. This change would permit the efficient layout of a majority of table designs while enabling sophisticated table designs to specify grid modules that were square or any other specified aspect ratio.

Additional typography capabilities will also be necessary to make the prototype fully functional. For example, the capability to center a set of aligned boxes within grid lines mentioned in Chapter 5.

Implementing a spreadsheet application based on the table formatting scheme would stress the performance of the implementation and reveal strategies for dealing with repetitive structures. It would also be an opportunity to implement automatic computation of carry-forward table entries, automatic insertion of continuation and summary headings, and various style attributes for improving the readability of large tables.

It would be desirable to accept a table design prespecified by a designer or precomputed from a prototypical table. Such a completely specified table layout should be recognized to avoid computing the dimensions of an immense number of table entries. The implementation would require strategies for dealing with table entries that do not conform to the prespecified design, at least by reporting them.

#### **6.4 Table Formatting Algorithms**

Determining an aesthetically pleasing table layout remains a significant research problem. The grid structure presented in this thesis provides a basis for dealing with the problem. Balancing the amount of whitespace within a table can be achieved by manipulating the position of grid lines. Some possible strategies might be to increase the column width while reducing the row height (or vice versa) or to fold a single long column into several shorter ones. A useful conjecture to prove empirically would be whether table entries have a row depth that increases monotonically with decreasing column width. The monotonic relationship would permit efficient search strategies. Some graphic design principles can be expressed in terms of constraints on grid lines, such as forcing table entries to be equal width or equal height when they are 'almost' equal.

We would like to investigate how to apply the table alignment techniques to mathematical notation and page layout. Many two-dimensional notations have positioning and alignment requirements similar to tables. For instance, matrix notation is obviously similar to a small table, and centered limits above a summation operator share a common alignment. When two or more summation notations appear in the same mathematical segment, we may wish that the corresponding parts are aligned together on a common grid line. Similarly, corresponding alignment would be desirable for large fractions with complex numerator and denominator expressions, for matrices with similar numbers of rows, and for integration operators with limits.

Page layout has long used a grid structure to describe the relationships among parts of the page. Page and table layout could share interactive graphic design tools for creating the grid structure and expressing the alignment relationships. Treating pages as a table layout would require defining a new kind of table entry for flowing text from one page to another, possibly related to the mechanisms involved in the spreadsheet system proposed in Section 6.3.

## 6.5 User Interface Design

The crucial requirement here is to develop an appropriate user model for interactively specifying graphic design intentions for layout and style. The specification of style sheets through the metaphor of programming macro commands for a document composition system is neither interactive nor intuitive. Interactive graphic design tools are needed to support the graphic designer to express their intentions within their own design framework.

The grid structure provides a metaphor common to the graphic design discipline. The interactive techniques implemented in NEWSWHOLE [Tilbrook, NEWSWHOLE] provide ideas for rubber-band grid lines, measurement rules, and gravity for nearby placement of objects. The use of a template to refer to a predefined table layout or illustration layout would simplify the construction of repeated designs.

The constraint system used in the prototype for table layout needs an interactive specification tool. Techniques are necessary for identifying alignment relationships among a set of grid lines. Special arrangements may have to be written in the expression language unless a suitable interactive specification technique can be developed.

## 6.6 Expert Style Rules

The application of expert systems and knowledge representations to the problem of document formatting are but a gleam in the researcher's eye. Capturing the style rules used in graphic design and book publishing may be a productive approach towards aesthetic and effective page layout. There are certainly many style manuals to choose from for the knowledge base. The grid structure used in the table formatting prototype described here may serve as a useful representation for the area layout problems that are the principal domain of graphic design.



# Glossary

---

Glossary terms are emphasized in an *italic sans-serif typeface* when they first appeared in the thesis. Readers viewing the glossary with Tioga can use the Tioga Def button to search for the definition of a selected glossary term. Select the term and middle-click Def to cause the glossary viewer to scroll to the definition of that term. For instance, select **reference mark** and middle-click Def to scroll immediately to its definition “**reference mark**: symbol that connects the reference information with its application in the table; for words, may use numbers, for numeric entries, use symbols [–, A Manual of Style, 1969, p 287].”

**acquisitions editor**: the person within a publishing organization who acquires a **manuscript** to be considered for publication.

**align on character**: **vertically align** several table entries on some character found within the entry; examples: decimal points, commas, dollar signs, percent signs, plus sign, minus sign,  
[–, A Manual of Style, 1969, p 284]

**artwork**: the manifestation of an illustration for a document; see **mechanicals**.

**asterisk**: the symbol \*, used as a reference mark.

**author**: the person who creates a **manuscript**.

**author's alterations**: modifications to a document requested by an **author** after composing the **manuscript**; a **publisher** commonly budgets about 10% of the cost for author's alterations, however each change is very expensive.

**back matter**: the parts of a publication that follow the main content; usually includes indices, glossaries, references, bibliographies, colophons; see **front matter**.

**baseline**: the line on which a string of characters sits.

**bearoff**: the whitespace between a table entry and the nearest **typographic rule** or neighboring table entry.

**blue lines**: lines drawn in a light blue color (pen or ink) on **camera-ready copy** that are invisible to the photographic process when preparing **printing plates**; useful for outlining the grid lines of a page design to make aligning material easier.

**body**: of a **table**, vertical **columns** to the right of the **stub** and below the **boxheads**.

**book editor**: the person who marks corrections within a **manuscript** to conform to a set of guidelines.

- boxes:** as in 'boxes-and-glue model' [Knuth, The TEXbook], the collection of rectangular areas that have a width, depth, and height; as in table entry boxes, the abstraction of a table cell as a rectangular area.
- boxhead:** the part of a table above the columns that usually contains the column headings; named for the use of **typographic rules** to completely enclose the headings with both horizontal and vertical rules.
- boxheading:** a synonym for boxhead, or one of the column headings within the boxhead.
- brace:** one of the pair of symbols {}, shows relationship of groups [—, A Manual of Style, 1969, p 285].
- breaking:** the action of separating joined parts of a word (by **hyphenation**), or of a line (by a **line-breaking algorithm**), or of a page (by a **pagination algorithm**).
- broadside table:** a table with its columns designed to run parallel to the short dimension of the page; see **vertical table** [—, A Manual of Style, 1969, p 273]
- brought forward:** a table entry at the beginning of a column of data copied from some preceding part of the table; see **carried forward**.
- buck tooth:** a word at the end of a line that dangles out over the line below it [van Leunen, Handbook, p 289].
- bullet:** the symbol •, used to mark the beginning of an item in a list; an open bullet has a dark ring around a light center, while a closed bullet is entirely dark.
- camera-ready copy:** the final form of a **document** prior to producing **printing plates**; frequently required to reduce production costs since the **author** must bear all the expense up to this stage.
- carried forward:** a table entry at the end of a column of data that is copied to some following part of a table; see **brought forward**.
- cell:** intersection of a row and **column** or **boxhead** and **stub**; a **table entry** [—, A Manual of Style, 1969, p 283].
- centered:** the positioning of an object to be equidistant between two points; common points are the left and right margins of a page, the left and right grid lines in a table column, or the top and bottom grid lines in a table row.
- class:** a collection of objects all of the same type; for example, the class of graphical objects that are lines.
- colophon:** a description of how the publication was produced; a publisher's emblem.
- column headings:** the identifying label that appears at the top of a **column**.
- column:** a collection of table entries arranged in a vertical stack.
- composition:** 1) literary composition, 2) document composition, 3) table composition; to make a whole out of parts.
- compositor:** the person who produces typeset material by composing **type** on a **typesetter**.
- constraint satisfaction:** the state of satisfying a set of constraints; usually a mathematical solution to a system of equations that describe the constraints.
- constraint solver:** an algorithm that computes a **constraint satisfaction** solution.
- constraints:** a rule that limits the behavior of an object.
- continued:** line of text indicating that tabular material continues from a previous part of the table [—, A Manual of Style, 1969, p 281].
- copy editor:** the person who marks corrections within a **manuscript** to conform to a set of guidelines; common guidelines include grammar rules, language usage, numbering schemes, and cross references.
- copy:** the textual material of a **manuscript** or **document**.
- copy fit:** the process of adjusting the formatting parameters to force **copy** to fill a particular area completely.
- craftsman:** a person who has developed considerable skill through practice and talent.
- cross relationships table:** items in the **stub** are identical to items in the **boxhead** [—, A Manual of Style, 1969, p 290].
- CRT:** cathode ray tube, the means of converting electronic signals to light by striking phosphor with an electron beam; used in computer displays.
- cut-in head:** a head that cuts across the body of a table and applies to the tabular

- material lying below [—, A Manual of Style, 1969, p 278, ex. p 279]
- dagger:** the symbol †, used as a reference mark.
- decimal point:** the symbol ., used to separate the whole and fractional parts of a number; used to align on character.
- decked heads:** two or more levels of boxheads [—, A Manual of Style, 1969, p 278, ex. p 279].
- difficult copy:** the kind of material in a manuscript or document that requires special or expensive handling; usually mathematical notation, tabular material, and foreign language material (including computer programming languages); also penalty copy.
- digital typesetter:** a device for producing type by electronic means, usually with a CRT or laser.
- ditto marks:** the symbol " (as distinguished from the double quotation mark symbol ") used within a table to indicate a repeated entry.
- docket:** an enclosing folder or envelope that contains all the parts of a publication during the production process; often the outside is used to record production steps, times, and expenses; see job docket.
- document:** a collection of information presented in a form to be read and understood by humans; traditionally documents are produced on paper, but the concept extends to an electronic form of presentation on a display screen.
- document compilers:** a computer program that formats a document from a specification of the content and a separate collection of formatting parameters.
- document models:** a collection of abstract representations of the organization of the content within a document; common models include a stream of text with interspersed formatting directives, and a hierarchical organization into sections of text.
- document structure:** the organization of the content within a document; typically a document has both a logical structure (the interrelationship of the content) and a physical structure (the organization of a document into pages or screens).
- document style:** a manner of formatting a document, perhaps guided by rules for headings, paragraphs, figures, captions, etc.
- double dagger:** the symbol ‡, used as a reference mark.
- double rule:** two typographic rules, perhaps of different thicknesses, separated by a small amount of whitespace.
- doubled up:** two halves of a table running side by side, with boxheads repeated over the second half [—, A Manual of Style, 1969, p 280].
- dummy:** a preliminary sample of a finished document, often produced to visualize the design of the document; may be only a crude approximation drawn by hand, or more elaborate.
- drop folio:** a page number or folio placed at the bottom of the page, typically for the opening page of a chapter or for a broadside table.
- editor:** 1) acquisitions editor, 2) copy editor, 3) journal editor, 4) production editor, 5) book editor, 6) text editor.
- elite:** a typewriter spacing system with 12 characters per inch; see pica.
- em space:** the distance equal to the point size of type, named for the letter 'M' which was often designed to be a square of that size.
- en space:** the distance equal to one-half an em space.
- flush bottom:** positioning so that the bottom of the item is at the extreme limit of vertical movement.
- flush left:** positioning so that the left of the item is at the extreme limit of horizontal movement.
- flush right:** positioning so that the right of the item is at the extreme limit of horizontal movement.
- flush top:** positioning so that the top of the item is at the extreme limit of vertical movement.
- foldout insert:** a table so wide that it must be printed separately on a large sheet of paper and bound into the book [—, A Manual of Style, 1969, p 274].
- folio:** a number identifying the page; the page number; see drop folio, running heads, running feet.

- footnote:** a portion of the document that should be read out of sequence to provide additional details; commonly used for scholarly references or explanations; may appear within tables to explain exceptional circumstances.
- format codes:** in TTS typesetting, similar to computer programming macros but named by numbers; typically there are a small number of them and each one has a fixed length.
- foundry type:** the cast metal type that are composed by hand into pages.
- front matter:** the parts of a publication that precede the main content; may include a half-title page, title page, imprint page, table of contents, table of figures, abstract, forward, and dedication; see **back matter**.
- furniture:** the various odd-sized pieces of used to fill voids in a page composed by hand from **foundry type**.
- galley:** the typeset form of part of a document, without any concern for the breaks between pages; typically a galley is about 14 inches long, although modern composition systems may produce galleys of unlimited length.
- galleys:** the collection of typeset forms for a complete document.
- genealogical table:** the ancestry of a human being, or the pedigree of an animal [—, *A Manual of Style*, 1969, p 290].
- gloss:** a brief explanatory note or translation of a difficult or technical expression, usually inserted in the margin or between lines of a **manuscript** or text [—, *Dictionary*].
- graphic artist:** the person who creates illustrations and artwork for a document.
- graphic arts:** the collection of skills and crafts in producing a printed document.
- graphic design:** the profession of creating designs for printed or visual material in the graphic arts.
- graphic designer:** a person who creates graphic designs.
- graphical style:** the application of document style techniques to the production of illustrations.
- grid system:** the disciplined use of lines to align parts of a document; created by a graphic designer.
- grid:** the rectangular array of lines in a grid system.
- gutter:** the whitespace between two columns of a table or a page.
- hairline rule:** a thin typographic rule, often 1/10 of a point thick.
- halftone:** a technique for reproducing images with varying densities by creating dots with a single density but varied size; an image produced through this technique.
- horizontal alignment:** see **horizontally aligned**.
- horizontally aligned:** the arrangement of two or more objects side-by-side along a common horizontal line; see **vertically aligned**.
- hot metal typesetting:** the process of mechanically producing slugs of type by casting molten metal (usually a lead alloy) into molds for each letterform; the slugs are assembled by hand into pages; after printing, the slugs could be melted and the metal reused.
- house style:** the appearance of books produced by a particular publishing house or company.
- hyphenation:** the separation of parts of a word into syl-la-bles by introducing hyphens.
- illustrator:** 1) a person who creates illustrations, 2) an interactive computer program that aids in creating illustrations.
- imposition:** the arrangement of several pages printed at once to reduce printing expenses and time; care is required to orient the pages so that after a **signature** is folded the pages are in the proper order and position.
- imprint page:** a page in the **front matter** that describes the publication; typically includes a copyright notice, address of the publisher, bibliographic information, and production information or credits.
- indexer:** person or computer program that produces the index entries
- italic:** a slanted typeface.
- jacket:** the cover of a book or document.
- job docket:** the record of the production history of a document; see **docket**.
- journal editor:** the person who collects and edits the contributions to a periodical journal.
- justification:** the process of making a line of text fill the measure.

- keyboarder:** the person who enters the typesetting codes for a **manuscript**.
- landscape:** the orientation with the short dimension vertical and long dimension horizontal; see **portrait**.
- leaders:** row of spaced periods [—, *A Manual of Style*, 1969, p 282] that “lead the eye.”
- leading:** the **whitespace** between adjacent lines of **type**; created by inserting a thin strip of lead.
- leading zeros:** the zero digits that precede a numeric value, for example, 03 or 000177.
- letterform:** the design or shape of a character in a **typeface**, usually an alphabetic letter.
- letterspace:** the space between letters in a word; the process of adjusting the space between letters to justify a line of **type**.
- line-breaking algorithm:** an algorithm that separates a long line of text into shorter lines equal to the measure; may involve **hyphenation** to separate long words, and either local or global considerations within a paragraph.
- ligatures:** the special design of symbol formed by the joining of two letters that often appear to touch, for example, the common ligatures for the letter ‘f’ are ff, fi, fi, ffi, ffi.
- lower-rail:** a TTS code for selecting an additional set of symbols in the typesetting device; see **upper-rail**.
- macro packages:** a collection of computer program macros that define formatting attributes and operations for a **document compiler**.
- manuscript:** the written form of a **document** prior to publication, often in hand-written or typewritten form.
- master images:** an image from which reproductions are made.
- mathematical composition:** the preparing of two-dimensional notation that contains mathematical symbols; typically considered **difficult copy**.
- measure:** the length of a line of **type**.
- mechanicals:** the production-quality **artwork** for an illustration.
- monospaced fonts:** all characters in a **typeface** are of equal width.
- moveable type:** the technique of printing by assembling the letters on a page from a collection of **type**; as opposed to hand-written originals or hand-carved plates.
- note:** remark that applies to the tabular material [—, *A Manual of Style*, 1969, p 287].
- number sign:** the symbol #, used as a **reference mark**.
- page breaks:** the separation of a **document** into pages; the places where pages are separated.
- page dummies:** **sample pages** that represent the finished **document**; see **dummy**.
- page layout:** the procedure of arranging all the components on a page.
- page repros:** abbreviation of **reproduction pages**.
- pagination:** the process of performing **page layout**.
- Painter’s algorithm:** an algorithm for rendering objects from back to front so the last object rendered is visible.
- paragraph mark:** the symbol ¶, used as a **reference mark**.
- parallels:** the symbol ||, used as a **reference mark**.
- peculiars:** characters which are additional to the normal font [Phillips, *Handbook*, p 188]; see **sorts**.
- penalty copy:** see **difficult copy**.
- phototypesetter:** a **typesetter** device that exposes each **letterform** through a CRT onto photographic material (film or paper).
- photomechanical transfer:** a photographic process for preparing **printing plates** that uses a large-format camera.
- pi character:** a symbol from a jumbled collection; often but not necessarily associated with a particular **typeface**.
- pi font:** a collection of **pi characters**, usually designed to accompany a particular **typeface**.
- pi sorts:** a jumbled collection of **pi characters**.
- pica:** a typesetting measurement equal to 12 points; a typewriter spacing system with 10 characters per inch; see **elite**.
- PMT:** abbreviation for **photomechanical transfer**.
- point:** a typesetting measurement equal to 0.013837 inch, approximately 1/72 of an inch; common unit for **type size**.

- portrait:** the orientation with the long dimension vertical and short dimension horizontal; see **landscape**.
- printing plates:** the metal, plastic, or paper sheet from which impressions are taken; see **signature**, **imposition**.
- production editor:** the person responsible for overseeing all the stages of producing a document.
- publisher:** the organization responsible for producing and disseminating a document.
- publishing house:** a reference to a publisher's establishment, no longer called houses.
- quadded:** the process of inserting space to justify a line of text.
- reference mark:** symbol that connects the reference information with its application in the table; for words, may use numbers, for numeric entries, use symbols [—, A Manual of Style, p 287].
- recto:** the right-hand page of two facing pages; see **verso**.
- reproduction pages:** the form of finished pages of a document ready for printing; see **camera-ready copy**, also **page repros**.
- row:** a collection of table entries arranged side-by-side.
- rule:** 1) a style rule, 2) a **typographic rule**.
- rules:** see **typographic rules**.
- running feet:** the identifying material at the bottom of each page in a chapter or section; may include the page number or **folio**.
- running heads:** the identifying material at the top of each page in a chapter or section; usually includes the page number or **folio**.
- runover:** another name for a **turnover** [—, A Manual of Style, 1969, p 281].
- sample pages:** pages produced to review the design of a document, the formatting specifications, or the capabilities of a formatting system; see **dummy**.
- sans-serif:** a **letterform** design without a serif; see **serif**.
- scabbard:** **whitespace**, half a line high, that surrounds a vertically centered heading with an even number of lines; like a knife sheath; to be avoided since the heading lines of adjacent columns will be on different baselines [—, GPO, p 189]
- section mark:** the symbol §, used as a reference mark.
- serif:** the short line or widening at the end of strokes in a **letterform** design; see **sans-serif**.
- shift-codes:** TTS codes for shifting between upper- and lower-case character codes.
- signature:** the impression from one **printing plate** that contains several pages; see **imposition**.
- single rule:** a single **typographic rule**; see **double rule**.
- small caps:** the **letterform** for the capital letters reduced in height; originally these were designs, distinct from the capital letters, that preserved the width of strokes, but they are now commonly produced on **phototypesetters** by reducing the type size of capital letters.
- solid:** positioning adjacent lines of text with no additional **leading** between them.
- sorts:** a collection of odd symbols in **foundry type**, typically tossed into a large tray; a symbol was found by sorting through the collection.
- source note:** credit placed at the bottom of a table indicating the source of the tabular data [—, A Manual of Style, 1969, p 286].
- space:** the distance between two objects; see **em space**, **en space**, **letterspace**, **wordspace**, **whitespace**.
- spacebands:** the adjustable-width spaces between words used to justify a line of **type**; originally, these were wedges mechanically expanded until the type fit the measure.
- star:** the symbol ★, used as a **reference mark**.
- stub:** first **column** of a table containing the **row** headings. [—, A Manual of Style, 1969, p 278].
- style:** a way of doing something. 1) **house style**, 2) **writing style**, 3) **document style**, 4) **graphical style**.
- subtotal:** sum of some values in a **column**.
- super-shift:** a TTS code for extending the meaning of the following code in a similar way that a shift-code makes a lower-case letter into an upper-case letter.
- table:** an orderly arrangement of information, typically as a rectangular array of rows and columns.
- table entry:** a cell at the intersection of a **row** and **column**.

- table parts:** boxhead, stub, panel.
- tabular composition:** the preparation of material arranged in a table; see **difficult copy**.
- text editor:** a computer program for modifying a manuscript stored as a computer file.
- total:** sum of values in a column.
- TTS:** tele-typesetting-system; a coding system, initially appropriate for 5-, 6-, 7-, or 8-level punched paper-tape devices, used for conveying both text content and formatting directives to typesetters.
- turnover:** text set on the second and subsequent lines of a table entry [—, *A Manual of Style*, 1969, p 281].
- type:** a small block of wood or metal that contains the raised design of a letter or symbol; a collection of printed or typewritten symbols.
- typeface:** a collection of symbols of a particular type family (Helvetica, Times Roman) in a common style (regular, italic), in a particular weight (light, medium, bold), with a particular spacing (condensed, expanded), and possibly at a particular size (10 points, 36 points); modern digital typesetters can create many variations of typefaces through electronic techniques (zooming, slanting, independent *x* and *y* scaling).
- typesetter:** a device for producing type; a person who operates a typesetting device; see **phototypesetter**, **digital typesetter**.
- typographic rule:** a thin line drawn horizontally or vertically within a document; usually to separate rows or columns within a table, or to separate different kinds of document content, for example, a figure and caption from the rest of a page.
- typography:** the art of composing printed material from moveable type.
- typographer:** person skilled in the art of typesetting text; may specialize in font design, type layout, lettering design.
- upper-rail:** see **lower-rail**; a TTS code for selecting an additional set of symbols in the typesetter.
- verso:** the left-hand or back page of two facing pages; see **recto**.
- vertical alignment:** see **vertically aligned**.
- vertically aligned:** the arrangement of two or more objects above one another along a common vertical line; see **horizontally aligned**.
- vertical rule:** a rule that is oriented vertically; awkward for typesetting systems that generate lines of type.
- vertical table:** a table with its columns designed to run parallel to the long dimension of the page; see **broadside table** [—, *A Manual of Style*, 1969, p 273].
- whitespace:** the space that contains no type or illustration; usually the margins surrounding a page or a table entry.
- wordspace:** the width of the normal space between words.
- writing style:** a manner of writing using particular spelling rules and grammar rules.
- WYSIWYG:** 'what you see is what you get' referring to displaying what will be printed; coined by Doug Engelbart; pronounced 'whizzy-wig.'

# References

- 
- [—, A Manual of Style, 1969] —. *A Manual of Style*. The University of Chicago Press, 12th edition, 1969.
- [—, APL/66] —. *APL/66 User's Guide*. Honeywell Information Systems, Series 60 (Level 66)/6000 DD78 Rev. 1, November 1977.
- [—, DCF] —. *Document Composition Facility User's Guide*. IBM Corporation, SH20-9161, April 1980.
- [—, Dictionary] —. *The American Heritage Dictionary of the English Language*. Morris, William, editor. Houghton Mifflin, 1978.
- [—, GML] —. *Document Composition Facility, Generalized Markup Language*. IBM Corporation, SH20-9188, April 1980.
- [—, Interpress] —. *Interpress Electronic Printing Standard*. Xerox System Integration Standard, XSIS 048404, April 1984.
- [—, JaM] —. *JaM Manual*. Unpublished manuscript, December 1984.
- [—, GPO] —. *U.S. Government Printing Office Style Manual*. Government Printing Office, 1973.
- [—, PostScript] —. *PostScript Language Manual*. Adobe Systems, First edition, revised, September 1984.
- [—, SCRIBE] —. *SCRIBE Document Production System*. UniLogic, Fourth Edition, April 1984.
- [—, TELE-A-GRAF] —. *TELE-A-GRAF User's Manual*. Integrated Software Systems, 1984.
- [—, The Chicago Manual of Style, 1982] —. *The Chicago Manual of Style*. The University of Chicago Press, 13th Edition, 1982.
- [AAUP, One Book/Five Ways] Association of American University Presses. *One Book/Five Ways*. William Kaufmann, 1978.
- [Alexander, Editor Aids] —. "Computer Aids for Authors and Editors," *The Seybold Report on Publishing Systems* 13 10. February 13, 1984, 3-18.
- [Alexander, Micros] George A. Alexander. "Micros in Publishing: The Push is On," *The Seybold Report on Publishing Systems* 13 16. May 14, 1984.
- [Alexander, Tyxset] George A. Alexander. "TEX on a Micro: The Tyxset Typesetting System," *The Seybold Report on Publishing Systems* 14 4. October 29, 1984, 9-14.
- [Allman, -me] Eric Allman. *-me Reference Manual*. Unpublished manuscript, 1983.



- [ANSI, GKS] ANSI Technical Committee X3H3 – Computer Graphics. “Graphical Kernel System,” *Computer Graphics*. Special GKS Issue, February 1984.
- [Aspvall&Shiloach, Linear Inequalities] Bengt Aspvall and Yossi Shiloach. “A Polynomial Time Algorithm for Solving Systems of Linear Inequalities with Two Variables per Inequality,” *SIAM Journal of Computing* 9 4. November 1980, 827-845.
- [Baker, 5/4 Algorithm] Brenda S. Baker, Donna J. Brown, and Howard P. Katseff. “A 5/4 Algorithm for Two-Dimensional Packing,” *Journal of Algorithms* 2 4. December 1981, 348-361.
- [Baker, Packings] Brenda S. Baker, E.G. Coffman Jr., and Ronald L. Rivest. “Orthogonal Packings in Two Dimensions,” *SIAM Journal of Computing* 9 4. November 1980, 846-855.
- [Barnett, Computer Typesetting] Michael P. Barnett. *Computer Typesetting: Experiments and Prospects*. MIT Press, 1965.
- [Baudelaire&Stone, Griffin] Patrick Baudelaire and Maureen Stone. “Techniques for Interactive Raster Graphics,” *Computer Graphics* 14 3. July 1980, 314-320.
- [Baudelaire, Draw] Patrick C. Baudelaire. “Draw Manual,” *Alto User's Handbook*. Xerox PARC, 1979, 97-128.
- [Beach&Stone, Graphical Style] Richard J. Beach and Maureen Stone. “Graphical Style – Towards High Quality Illustrations,” *Computer Graphics* 17 3. July 1983, 127-135.
- [Beach, Computerized Typesetting] Richard J. Beach. *Computerized Typesetting of Technical Documents*. University of Waterloo Computer Science Technical Report CS-77-38, June 1977.
- [Beach, CS740 project] Richard J. Beach. *A Data Structure for Formatting Hierarchical Tables*. Unpublished manuscript, December 1983.
- [Beach, Paint] Richard J. Beach, John C. Beatty, Kellogg S. Booth, Eugene I. Fiume, and Darlene A. Plebon. “The Message is the Medium: Multiprocess Structuring of an Interactive Paint Program,” *Computer Graphics* 16 3. July 1982, 277-287.
- [Beach, PROFF] Richard J. Beach. *Photon ROFF Text Formatter (PROFF)*. University of Waterloo, Computer Science Research Report, CS-76-08, April 1976.
- [Beach, Typeset] Richard J. Beach. *Type Reference Manual*. Unpublished manuscript, 1981.
- [Beatty, Picture] John C. Beatty, Janet S. Chin, and Henry F. Moll. “An Interactive Documentation System,” *Computer Graphics* 13 2. August 1979, 71-82.
- [Bell, Sc.Am. illustration] Edward Bell. Personal communication concerning the many iterations needed by illustrators to meet the *Scientific American* style and quality expectations. 1983.
- [Berg, Composition] N. Edward Berg. *Electronic Composition*. Graphic Arts Technical Foundation, 1975.
- [Biggerstaff, TABLE] Ted Biggerstaff, D. Mack Endres, and Ira R. Forman. “TABLE: Object Oriented Editing of Complex Structures,” *IEEE Conference on Software Productivity*. 1984, 334-344.
- [Booth&Gentleman, Anthropomorphism] K.S. Booth, W.M. Gentleman and Jonathan Schaeffer. “Anthropomorphic Programming,” Conference on Issues for Large Scale Computing. Salishan Lodge, Oregon, March 1982. Also available as University of Waterloo Computer Science Research Report, CS-82-47, February, 1984.

- [Borning, Thinglab] Alan Borning. *ThingLab — A Constraint-Oriented Simulation Laboratory*. PhD thesis, Stanford University, 1979. Also available as Xerox PARC Technical Report SSL-79-3, 1979 or Stanford Computer Science Department Report STAN-CS-79-746, July 1979.
- [Brader, Incremental Formatter] Mark Stuart Brader. *An Incremental Text Formatter*. University of Waterloo, Computer Science Research Report, CS-81-12, April 1981.
- [Cargill, Views] T.A. Cargill. *A View of Source Text for Diversely Configurable Software*. University of Waterloo, Computer Science Technical Report, CS-79-28, 1979.
- [Carmody, Hypertext] S. Carmody, W. Gross, T.E. Nelson, D. Rice and A. van Dam. "A Hypertext Editing System for the /360," in *Pertinent Concepts in Computer Graphics* [Faiman, Pertinent Graphics]. 1969.
- [Chamberlin, JANUS] D.D. Chamberlin, *et al.* "JANUS: An interactive document formatter based on declarative tags," *IBM Systems Journal* 21 3. 1982, 250-271.
- [Cherry, Writing Tools] Lorinda L. Cherry. "Writing Tools," *IEEE Transactions on Communications* 1. January 1982, 100-105.
- [Coffman, 2D Packing] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, and R.E. Tarjan. "Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms," *SIAM Journal of Computing* 9 4. November 1980, 808-826.
- [Corliss&Bozman, NBS53] C.H. Corliss and W.R. Bozman. "Experimental Probabilities for Spectral Lines of Seventy Elements," *NBS Monograph* 53. July 20, 1962.
- [Crow, Image Environment] F.C. Crow. "A More Flexible Image Generation Environment," *Computer Graphics* 16 3. July 1982, 9-18.
- [Dantzig, LP] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.
- [Davis, Tabular Presentation] James A. Davis and Ann M. Jacobs. "Tabular Presentation," *International Encyclopedia of the Social Sciences*. 1968, 497-509.
- [Dyck, Computing] V.A. Dyck, J.D. Lawson, J.A. Smith, and R.J. Beach. *Computing: An Introduction to Structured Problem Solving Using PASCAL*. Reston, 1982.
- [Dyck, FORTRAN77] V.A. Dyck, J.D. Lawson, and J.A. Smith. *FORTAN 77: An Introduction to Structured Problem Solving*. Reston, 1983
- [Dyck, PASCAL] V.A. Dyck, J.D. Lawson, J.A. Smith, and R.J. Beach. *Computing: An Introduction to Structured Problem Solving Using PASCAL*. Reston, 1982
- [Dyck, WATFIV-s] V.A. Dyck, J.D. Lawson, and J.A. Smith. *Introduction to Computing: Structured Problem Solving Using WATFIV-S*. Reston, 1979
- [Dyment, Corkscrew] Doug Dyment. "A Corkscrew for the Software Bottleneck," *Micros* 1 2. October 1982, 21-24.
- [Engelbart, NLS] Douglas C. Engelbart and William K. English. "A research center for augmenting human intellect," *AFIPS Conference Proceedings* 33. 1968, 395-410.
- [Engelbart, Terminals] Douglas C. Engelbart. "Design considerations for knowledge workshop terminals," *Proceedings of the National Computer Conference*. June 1973, 221-227.
- [Faiman, Pertinent Graphics] M. Faiman and J. Nievergelt (eds). *Pertinent Concepts in Computer Graphics*. University of Illinois, 1969.
- [Furuta, Survey] Richard Furuta, Jeffery Scofield, and Alan Shaw. "Document Formatting Systems: Survey, Concepts, and Issues," *Computing Surveys* 14 3. September 1982, 417-472.

- [Garey&Johnson, NP] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [George&Liu, Sparse Matrices] Alan George and Joseph W-H. Liu. *Computer Solution of Large Sparse Positive Definite Matrices*. Prentice-Hall, 1981.
- [Goines, Alphabet] David Lance Goines. *A Constructed Alphabet*. David R. Godine, 1982.
- [Good, Etude interface] Michael. Good. "Etude and the Folklore of User Interfaces," *SIGPLAN Notices* 16 6. June 1981, 34-43.
- [Gruhn, YFL] Ann M. Gruhn. *The Yorktown Formatting Language*. IBM Computer Science Research Report, RC6994, June 1, 1978.
- [Hall, Tabular Presentation] Ray Ovid Hall. *Handbook of Tabular Presentation*. Ronald Press, New York, 1943.
- [Hammer, Etude] Michael Hammer, *et al.* "The implementation of Etude, an integrated and interactive document production system," *SIGPLAN Notices* 16 6. June 1981, 137-141.
- [Hurlburt, The Grid] Allen Hurlburt. *The Grid*. Van Nostrand Reinhold, 1978.
- [Ilson, Etude] Richard Ilson. *An Integrated Approach to Formatted Document Production*. Masters thesis, Massachusetts Institute of Technology, August 1980.
- [Johnson, CACM] Stephen C. Johnson. Private communication concerning typesetting journal articles within Bell Laboratories, 1984.
- [Johnson, JACM style] Stephen C. Johnson. Personal communication concerning the difficulties when typesetting journal articles for JACM. 1984.
- [Kahn&Hewitt, Actors] Kenneth M. Kahn and Carl Hewitt. "Dynamic Graphics Using Quasi Parallelism," *Computer Graphics* 12 3. August 1978, 357-362.
- [Kernighan&Cherry, eqn] Brian W. Kernighan and Lorinda Cherry. "A System for Typesetting Mathematics," *Communications of the ACM* 18 3. March 1975, 151-157.
- [Kernighan, ditroff] Brian W. Kernighan. *A Typesetter-independent TROFF*. Bell Laboratories Computing Science Technical Report, 97, March 1982.
- [Kernighan, pic] Brian W. Kernighan. "PIC—A Language for Typesetting Graphics," *Software—Practice and Experience* 12 1. January 1982, 1-21.
- [Khachiyan, LP] L.G. Khachiyan. "A Polynomial Algorithm in Linear Programming," *Doklady Akademii nauk SSSR Novaia Seriya* 244. 1979, 1093-1096. [English translation in Soviet Mathematics Doklady 20, 1979, 191-194.]
- [Kimura, thesis] Gary D. Kimura. *A Structure Editor and Model for Abstract Document Objects*. PhD thesis, University of Washington, July 1984. Also available as Technical Report 84-07-04, Department of Computer Science, University of Washington.
- [Kimura&Shaw, Abstract Documents] Gary D. Kimura and Alan C. Shaw. "The Structure of Abstract Document Objects," *SIGOA Newsletter* 5 1-2. June 1984, 161-169.
- [Knuth, AMS lecture] D.E. Knuth. "Mathematical typography," *Bulletin (New Series) of the American Mathematical Society* 1 2. March 1979, 337-372. Reprinted in *TEX and METAFONT: New Directions in Typesetting*. American Mathematical Society and Digital Press. 1979, Chapter 1.

- [Knuth, Line Breaking] Donald E. Knuth and Michael F. Plass. "Breaking Paragraphs into Lines," *Software—Practice and Experience* 11. 1981, 1119-1184.
- [Knuth, METAFONT] D.E. Knuth. "METAFONT, a system for alphabet design," *TEX and METAFONT: New Directions in Typesetting*. American Mathematical Society and Digital Press, 1979, Chapter 3.
- [Knuth, The TEXbook] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.
- [Lamport, L<sup>a</sup>TEX] Leslie Lamport. *The L<sup>a</sup>TEX Document Preparation System*. Unpublished manuscript, February 11, 1984.
- [Lampson, Bravo] Butler W. Lampson. "Bravo Manual," *Alto User's Handbook*. Xerox PARC, 1979, 31-62.
- [Leith, Metal type] Robert Leith. Private communication concerning typesetting customers insisting on metal type for legal documents to avoid the potential for errors in electronic compositions systems using phototypesetters, 1981.
- [Lesk, -ms] M.E. Lesk. *Typing Document on the UNIX System: Using the -ms macros with Troff and Nroff*. Bell Laboratories Internal Memorandum, October 8, 1976.
- [Lesk, refer] M.E. Lesk. *Some Applications of Inverted Indexes on the UNIX System*. Bell Laboratories Computing Science Technical Report, 69, June 1978.
- [Lesk, tbl] M.E. Lesk. *Tbl - A Program to Format Tables*. Bell Laboratories Computing Science Technical Report, 49, September 1976.
- [Liang, Hyphenation] Frank Liang. "Word Hy-phen-a-tion by Com-put-er," PhD thesis, Department of Computer Science, Stanford University, 1983. Also available as Stanford Computer Science Department Report STAN-CS-83-977, 1983.
- [Lipkie, Star Graphics] Daniel E. Lipkie, Steven R. Evans, John K. Newlin, and Robert L. Weissman. "Star Graphics: An Object-Oriented Implementation," *Computer Graphics* 16 3. July 1982, 115-124.
- [Lusignan, ICCH3] Serge Lusignan and John S. North (eds). *Computing in the Humanities: Proceedings of the Third International Conference on Computing in the Humanities*. University of Waterloo Press, 1977.
- [Macdonald, Writer's Workbench] Nina H. Macdonald, Lawrence T. Frase, Patricia S. Gingrich, and Stacey A. Keenan. "The Writer's Workbench: Computer Aids for Text Analysis," *IEEE Transactions on Communications* 1. January 1982, 105-110.
- [Monier, Scribe math] Louis Monier. Private communication concerning implementing mathematics formatting for Scribe, 1984.
- [Müller-Brockman, Grid Systems] Josef Müller-Brockman. *Grid systems in graphic design*. Hastings House Publishers, 1981.
- [Myers, Incense] Brad A. Myers. "Incense: A System for Displaying Data Structures," *Computer Graphics* 17 3. July 1983, 115-125.
- [Knott, Napier commemorative] Cargill Gilston Knott, editor. *Napier Tercentenary Memorial Volume*, Published for the Royal Society of Edinburgh by Longmans Green and Co., London, 1915.
- [Nelson, Juno] Greg Nelson. "Juno, a constraint-based graphics system," *Computer Graphics* 19 3. July 1985.
- [Nelson, Program Verification] Greg Nelson. *Techniques for Program Verification*. Xerox PARC Technical Report CSL-81-10, 1981.

- [Nelson, Literary Machines] Ted Nelson.  
*Literary Machines*. Ted Nelson,  
Swarthmore, PA, 1981.
- [Nelson, Xanadu] Theodor H. Nelson.  
"Replacing the Printed Word: A  
Complete Literary System," *Information  
Processing 1980, IFIPS*. 1980,  
1013-1023.
- [Newman&Sproull, Computer Graphics]  
W.M. Newman and R.F. Sproull.  
*Principles of Interactive Computer  
Graphics*, Second edition. McGraw Hill,  
1973.
- [Newman, Markup] William M. Newman.  
"Markup Manual," *Alto User's  
Handbook*. Xerox PARC, 1979, 85-96.
- [Ossanna, troff] Joseph F. Ossanna.  
*NROFF/TROFF User's Manual*. Bell  
Laboratories Computing Science  
Technical Report, 54, October 1976.
- [Ousterhout, Corner Stitching] John K.  
Ousterhout. *Corner Stitching: A Data  
Structure Technique for VLSI Layout  
Tools*. University of California,  
Berkeley, Computer Science Division  
Report UCB/CSD82/114, December  
1982.
- [Peck, Chaucer] Russell A. Peck (ed).  
*Chaucer's Lyrics and Anelida and Arcite*.  
University of Toronto Press, 1983.
- [Phillips, Computer Typesetting] Arthur  
Phillips. *Computer Peripherals and  
Typesetting*. Her Majesty's Stationery  
Office, 1968.
- [Phillips, Handbook] Arthur H. Phillips.  
*Handbook of Computer-Aided  
Composition*. Marcel Dekker, 1980.
- [Phillips, Tabular Composition] Arthur H.  
Phillips. "Tabular Composition," *The  
Seybold Report* 8 23. August 13, 1979.
- [Pierson, PAGE-1] John Pierson. *Computer  
Composition using PAGE-1*.  
Wiley-Interscience, 1972.
- [Plass, Optimal Pagination] Michael F.  
Plass. *Optimal Pagination Techniques for  
Automatic Typesetting Systems*. Xerox  
PARC Technical Report, ISI-81-1,  
August, 1981.
- [Plass, pagination] Michael F. Plass.  
*Optimal Pagination Techniques for  
Automatic Typesetting Systems*. Xerox  
PARC Technical Report, ISI-81-1,  
August, 1981.
- [Reid, Scribe thesis] Brian K. Reid. *Scribe:  
A Document Specification Language and  
its Compiler*. PhD thesis, Carnegie  
Mellon University, October 1980.
- [Ritchie, Turing Lecture] Dennis M.  
Ritchie. "Reflections on Software  
Research," 1983 ACM Turing Award  
Lecture, *Communications of the ACM* 27  
8. August 1984, 758-760.
- [Rosenthal, Graphical Resources] David  
S.H. Rosenthal. "Managing Graphical  
Resources," *Computer Graphics* 17 1.  
January 1983, 38-45.
- [Saltzer, RUNOFF] J. Saltzer. "Manuscript  
typing and editing: TYPSET, RUNOFF,"  
*The Compatible Time-Sharing System: A  
programmer's guide*. MIT Press, 1965,  
Section AH.9.01.
- [Seybold, Fundamentals] John W. Seybold.  
*Fundamentals of Modern  
Photocomposition*. Seybold Publications,  
1979.
- [Seybold, MacWrite] Seybold, Jonathan.  
"Macintosh Publishing Systems," *The  
Seybold Report on Publishing Systems* 14  
9. January 28, 1985.
- [Seybold, Xerox's Star] Jonathan Seybold.  
"Xerox's Star," *The Seybold Report* 10  
16. April 27 1981.
- [Shand, CornerStitching] Mark Shand.  
*CornerStitching*. Unpublished  
manuscript, December 1984.
- [Smith, Star Interface] David Canfield  
Smith, Charles Irby, Ralph Kimball, and  
Eric Harslem. "The Star User Interface:  
An Overview," *Proceedings of National  
Computer Conference*. June 1982,  
515-528.

- [Stevens, NBS99] Mary Elizabeth Stevens and John L. Little. "Automatic Typographic-Quality Typesetting Techniques: A State-of-the-Art Review," *NBS Monograph 99*. April 7, 1967.
- [Strachey, GPM] C. Strachey. "General-Purpose Macrogenerator," *Computer Journal*. October 1965.
- [Sutherland, Sketchpad] I.E. Sutherland. "SKETCHPAD: A Man-Machine Graphical Communication System," *SJCC*. 1963, 329.
- [Teitelman, Cedar] Warren Teitelman. *The Cedar Programming Environment: A Midterm Report and Examination*. Xerox PARC Technical Report, CSL-83-11, June 1984.
- [Thomas, Graphics Parameters] Elaine L. Thomas. "Methods for Specifying Display Parameters in Graphics Programming Languages," *ACM SIGPLAN/SIGGRAPH symposium on Graphic Languages*. April 26-27, 1976, 54-56.
- [Tilbrook, NEWSWHOLE] David M. Tilbrook. *A Newspaper Pagation System*. Masters thesis, Computer Science Department, University of Toronto, 1976.
- [Updike, Printing Types] Daniel Berkeley Updike. *Printing types, their history, forms, and use: a study in survivals*, 2nd edition. Harvard University Press, 1937. Reprinted by Dover, 1980.
- [van Leunen, Handbook] Mary-Claire van Leunen. *A Handbook for Scholars*. Alfred A. Knopf, 1978.
- [van Leunen, One Document] Mary-Claire van Leunen. *How I Wrote One Document*. Unpublished manuscript, February 2, 1984.
- [van Wyk, ideal] Christopher J. van Wyk. "A Graphics Typesetting Language," *SIGPLAN Notices 16 6*. June 1981, 99-107.
- [Warnock&Wyatt, CedarGraphics] John Warnock and Douglas K. Wyatt. "A Device Independent Graphics Imaging Model for Use with Raster Devices," *Computer Graphics 16 3*. July 1982, 313-319.
- [White, Picc] Alex R. White. *Picc - A C-based Illustration Language*. Masters thesis, Department of Computer Science, University of Waterloo, 1981.
- [Williamson, Book Design] Hugh Williamson. *Methods of Book Design*. Oxford University Press, 1966.
- [Winograd&Paxton, T<sub>E</sub>X Indexing] Terry Winograd and Bill Paxton. "An Indexing Facility for T<sub>E</sub>X", *Tugboat (T<sub>E</sub>X users' group newsletter)*, 1 1, October 1980, Appendix A.
- [Witten, Traps] Ian H. Witten, Mike Bonham, and Evelyne Strong. "On the Power of Traps and Diversions in a Document Preparation Language," *Software - Practice and Experience 12*. 1982, 1119-1131.
- [Zabula-Salelles, GOB] Ignatio Andres Zabula-Salelles. *Interacting with Graphic Objects*. Stanford Computer Science Department Report, STAN-CS-82-960, December 1982.
- [Zeisel, Figures] Hans Zeisel. *Say It With Figures*. Harper & Row, Fourth edition, 1957.

# Index

---

## A

- abstract document object model, 2-39, 6-2
- acquisition editor, 2-8, G-1
- Adobe Systems Inc., 2-14
- aesthetics, 1-9, 4-2, 4-4, 4-7, 4-19, 5-5, 5-8, 5-20, 6-3, 6-5
- Alexander, George A., 2-8, 2-26, 2-32, R-1
- algorithm
  - dynamic programing, 2-31
  - enumerate area, 5-30, 5-31–5-32, 5-33, 5-45
  - graphics rendering, 3-15, 3-16
  - hit testing, 5-32–5-33
  - hyphenation, 2-30
  - incremental formatting, 2-32
  - Janus packer, 2-35
  - line breaking, 2-30, 2-31, 2-34, 3-5, 5-19
  - minimal repainting, 6-2–6-3
  - optimal page breaking, 2-30, 2-31, 4-14
  - page breaking, 2-21
  - painter's, 5-29
  - Simplex method, 5-37
  - table entry deletion, 5-32, 5-33–5-35, 6-2
  - table entry insertion, 5-32, 5-35, 6-2
  - table layout, 5-30, 5-32, 5-44–5-45
  - table rendering, 5-30, 5-32
- alignment, 4-7
  - alignment (continued)
    - constraints, 5-38
    - decimal point, 4-3, 4-8, 5-8, 5-16, 5-17, 5-19, G-2
    - flush bottom, 3-16, 4-8–4-9, 5-16–5-18, G-3
    - flush left, 3-16, 4-8, 5-16–5-18, 5-39, G-3
    - flush right, 3-16, 4-8, 5-16–5-18, 5-39, G-3
    - flush top, 3-16, 4-8–4-9, 5-16–5-18, G-3
    - horizontal, 4-7, 5-11, 5-17, 5-37, G-4
    - mathematical equations, 4-8
    - point, *see* alignment point
    - tab stops, 4-17
    - top baseline, 5-16, 5-18, 5-24
    - vertical, 4-7, 4-8, 5-11, 5-37
    - whitespace balancing, 5-19
  - alignment point, 5-9, 5-11, 5-14, 5-27, 5-30, 5-37, 5-39
- Allman, Eric, 2-26
- AMS lecture by Knuth, 2-31
- ANSI, 3-6, R-2
- anthropomorphism, 2-6
- APL/66, 1-6, R-1
- archiving tools, 1-5
- ARPANET, 2-29
- arrows, 3-3, 3-17
- artist procedure, 5-9
- ArtworkClass property, 3-9–3-12, 3-19, 5-21
  - ArtworkFileName, 3-12
  - ArtworkImage, 3-12
  - ArtworkNode, 3-10
  - ArtworkPath, 3-12
  - Table, 5-21

Aspvall, Bengt, 5-37, R-2  
 author's alteration, 1-3, 2-38, G-1  
 author, 1-2, 2-2, 2-4, 2-7, 2-8, 2-10, 2-11, 2-22,  
 2-27, 2-38–2-39, 3-2, G-1

## B

Bézier parametric cubic curves, 3-12, 3-13  
 back matter, 2-3, G-1  
 background tints, 4-12, 4-13, 5-2, 5-22, 5-23,  
 5-24, 5-27, 5-28, 5-31, 5-46  
 backslash character, 2-20, 2-24  
 Baker, 5-4, 5-6, R-2  
 balancing whitespace, 5-19, 5-20, 5-40, 5-43,  
 6-3  
 Barnett, Michael P., 2-19, 4-3, R-2  
 baseline alignment, *see* baseline alignment  
 Baskerville typeface, 2-18, 2-38  
 batch processing, 1-8, 4-19  
 Baudelaire, Patrick, 3-5, R-2  
 Beach, Richard J., 1-5, 1-9, 2-6, 2-19, 2-20,  
 2-21, 2-28, 3-1, 3-17, 3-18, 5-12, R-2  
 bearoff distance, 4-10, 4-13, 5-3, 5-16, G-1  
 Beatty, John C., xi, 3-5, R-2  
 Bell, Edward, 1-10, R-2  
 Bell Laboratories, 2-23, 2-26  
 Berg, N. Edward, 2-19, R-2  
 bibliographic references, 2-25, *see also*  
*troff*, *refer*  
 Biggerstaff, Ted, 2-27, 4-19, 4-20, R-2  
 BIN PACKING PROBLEM, 5-5  
 binder, 2-14  
 binding gutter, 4-15  
 bitmap illustrator, 3-5  
 blank lines, 4-4  
 blue and white technical reports, 3-9  
 blue lines, 2-13, G-1  
 bold typeface, 4-10, 5-17  
 book designer, 3-3  
 Booth, Kellogg S., xi, 2-6, R-2  
 border patterns, 3-17, 4-12, 6-2  
 Borning, Alan, 5-35–5-36, R-3  
 bounding box, 3-16, 3-18  
 box head, 4-4, 4-5, 4-16, 4-18, G-2  
 boxes and glue model, 2-30, 2-31, 4-19, 5-9,  
 G-2  
 Bozman, W.R., 2-19, 4-3, R-3  
 braces within tables, 4-12  
 Brader, Mark S., xii, 2-20, 2-24, R-3  
 Bravo text editor, 2-37, R-5  
 breaking tables, 4-14

broadside tables, 4-15, G-2  
 brought forward totals, 4-15, 6-3, G-2  
 business graphics, 3-5

## C

C programming language, 3-5  
 Cargill, Tom, 2-18, 2-40, R-3  
 Carlson, Chester, 5-10  
 Carmody, S., 1-4, R-3  
 carried forward totals, 4-15, 6-3, G-2  
 Cedar, 1-2, 2-17, 2-37, 3-1, 3-7, 3-12, 5-20,  
 5-25, 6-1, R-7  
 CedarGraphics, 3-1, R-7, *see also* device  
 independent graphics package  
 centering, 4-16, G-2  
 Chamberlin, David, 2-35, R-3  
*Chaucer's Lyrics and Anelida and Arcite*, 1-5,  
 R-6  
 chemical formulae, 1-4  
 Cherry, Lorinda L., 2-5, 2-25, 2-28, R-3, R-4  
*Chicago Manual of Style, The*, 1-10, 2-2, 2-5,  
 4-2, 4-11, R-1  
 coder, 2-12  
 Coffman, 5-6, 5-7, R-3  
 color printing, 2-14, 2-18, 6-2  
 color separations, 2-14  
 column headings, vertical, 4-15  
 column hierarchy, 5-12–5-13  
 column structure, 4-4, 5-2  
 column style, 4-10  
 column width, 4-4  
 command character, 2-24  
 commercial typesetting, 2-21, 2-26  
*Communications of the ACM*, 2-26, 5-2  
 complexity of table formatting, 5-3–5-9  
 compositor, 2-11, 2-12, 2-15, 2-16  
 compressed typeface, 4-13  
 Computer Graphics Laboratory, xi  
 computer programs, typesetting of, xii, 1-6,  
 2-8, 3-2  
 computerized typesetting, 1-9, 2-19  
*Computing*, 3-3, R-3  
 consistency among illustrations, 2-12, 3-1, 3-4,  
 3-19  
 constraint satisfaction, 5-16, G-2  
 constraint solver, 2-31, 5-1, 5-14, 5-30,  
 5-37–5-42, 6-1, 6-4, G-2  
 linear constraints, 5-36  
 nonlinear constraints, 5-36  
 objective function, 5-38, 5-40



constraints, in table prototype, 5-22  
 continuation headings, 4-14–4-15, 4-18  
 copy editor, 2-2, 2-4, 2-8, 2-10, 2-11, 2-15,  
 2-16, G-2  
 copy fit, 2-12, G-2  
 Corliss, C.H., 2-19, 4-3, R-3  
 corner stitching, 5-24–5-28, 6-2, R-6  
   Cedar implementation, 5-26, 5-32  
   coordinate system, 5-26  
   data structure, 5-24–5-28  
   enumeration algorithm, 5-31  
   neighbor finding algorithm, 5-32  
   point finding algorithm, 5-32  
   tesselations, 5-26, 5-27  
   tiles, 5-26, 5-27  
 corrections, 1-8  
 cross references, 1-3, 1-4, 1-8, 2-10, 6-2  
 Crow, Frank, 3-5, 3-19, R-3

## D

Dantzig, G., 5-37, R-3  
 data structure, 2-24, 2-28, 2-35, 2-40, 3-2, 5-2  
 database, 1-4, 2-29, 2-30, 2-40, 4-3  
 DCF (Document Composition Facility), 2-35,  
 R-1  
 debugging, 2-23  
 DEC PDP-11, 2-27  
 decimal point alignment, *see* alignment,  
   decimal point  
 decorations, 5-2, 5-14, *see also* ornaments  
 delimiters, 2-24, 2-25  
 design, 2-22  
 device resolution, 2-18, 2-38, 5-11, 5-31  
 device-independent graphics package, 5-11,  
 5-30  
 device-independent troff, 2-24  
 diction analysis, 1-7, 2-11  
 difficult copy, 2-5  
 display screen, 2-18, 2-23, 2-38, 3-6, 5-11,  
 5-31, 5-32  
 displayed equations, 2-25  
 ditroff, 2-24  
 diversions, 2-27, 2-28, *see also* traps  
 document compilers, 1-9, 1-10, 2-22, 2-23,  
 2-24, 2-28, 2-29, 2-35–2-37, 2-38, 3-1, 3-3,  
 3-6, 3-9, G-3  
 Document Composition Facility (DCF), 2-35  
 document formatter, 2-20, 2-22, 2-25, 2-30,  
 2-31, 3-4  
 document interchange, 5-21

document layout, 2-31  
 document models, 2-1, 2-39, 5-46, 6-1, G-3  
 document object, 3-10, 5-3, 5-9, *see also*  
   layout procedure, and rendering procedure  
 document structure, 2-34, 2-39, 5-3, G-3  
 document style, 1-7, 1-9–1-10, 2-22, 2-26,  
 5-1, G-3  
 documents, 1-2, 2-22, G-2  
   electronic, 1-1  
 dot leaders, *see* leaders  
 draft manuscripts, 2-8, 2-9, *see also*  
   manuscripts  
 draftsman, 1-8, 3-2, 3-3  
 Draw illustrator, 3-5  
 drop shadows, 3-17–3-18  
 Dyck, V.A., xii, 1-5, 3-2, 3-3, R-3  
 Dymont, Doug, 2-6, R-3  
 dynamic programming, 5-5

## E

editing, 2-24, 2-35, 2-37, 2-40, 3-6  
 editor, G-3  
   acquisition, 2-8  
   aids, 2-8  
   Bravo, 2-37, R-5  
 editor (continued)  
   copy, *see* copy editor  
   journal, 2-8  
   production, 2-2, 2-4, 2-8, 2-9  
   text, 2-8  
   Tioga, *see* Tioga  
 electronic composition tools, 1-2, 1-5  
   lack of integration, 1-5, 2-5  
 electronic documents, 1-1, 5-1  
 Engelbart, Douglas C., 1-4, 2-20–2-21, 2-32,  
 2-40, 3-7, R-3  
 eqn, 1-6, 1-7, 2-25, 2-26, 3-2, 4-18, 4-19, 5-9,  
 5-17  
   neqn, 2-26  
   syntax checker, 2-23  
 equal width columns, 4-7, 4-9, 5-20, 5-22,  
 5-40, 6-3  
 equation solver, *see* constraint solver  
 Etude, 2-1, 2-34, 2-35, 2-39, 3-9, R-4  
 expert systems, 1-10, 6-1, 6-5

## F

facing pages, 2-16, 4-15  
 figure caption, 4-6

filter/pipe model, 2-25  
 final pages, 1-8, *see* page repros  
 fixed width characters, *see* monospaced fonts  
 flowing between table entries, 4-6  
 flush bottom alignment, *see* alignment, flush  
 bottom  
 flush left alignment, *see* alignment, flush left  
 flush right alignment, *see* alignment, flush  
 right  
 flush top alignment, *see* alignment, flush top  
 folding table entries, 4-6, 4-9, 4-14, 4-17,  
 4-18, 5-17, 5-20, 5-43, 6-3  
 foldout plate, 4-15, G-3  
 footnotes, 1-8, 2-16, 2-18, 2-27, 6-2, G-4  
 in tables, 4-7, 4-10, 4-12–4-13  
 foreign languages, xii, 1-4, 2-10  
 form *versus* content, 1-1, 1-10, 2-29, 2-30,  
 2-34, 3-7  
 format codes, 2-12, 2-15, G-3  
 formatting commands, 1-5, 2-22, 2-36, 2-39,  
 3-6, 3-9  
 formatting rule, 4-6  
 formatting tags, , 2-22, 2-23, 2-24, 2-36  
 FORTRAN77, 1-5, R-3  
 foundry type, 2-13, G-3  
 fractions, 2-26  
 front matter, 2-3, 2-8, G-3  
*Fundamentals of Modern Photocomposition*,  
 2-19, 2-21  
 furniture, 2-13, G-3  
 Furuta, Richard, 1-9, 2-18, 2-20, 2-24, R-3

## G

galley, 1-3, 2-8, 2-21, G-3  
 Garamond typeface, 2-18, 2-38  
 Garey, 5-4, 5-5, R-4  
 Generalized Markup Language, *see* GML  
 Gentleman, Morven W., xii, 2-6, R-2  
 geometric representation, 3-12–3-15, 3-17,  
 3-18  
 geometry, 5-3  
 George, Alan, 1-7, 3-2, R-4  
 GKS (Graphical Kernel System standard),  
 3-6, R-2  
 gloss, 4-10, G-3  
 glue, 3-10  
 GML (Generalized Markup Language), 2-35,  
 R-1  
 GOB illustrator, 3-5, 6-2, R-7  
 Goines, David Lance, 5-15, R-4

Good, Michael, 2-34, R-4  
 GPM (General Purpose Macroprocessor), 2-28,  
 R-7  
 graphic artists, 1-3, 3-20, G-3, *see also* graphic  
 designer  
 graphic arts, 1-3, 2-28, 2-35 2-40, G-3  
 standards of quality, 1-3, 1-4, 1-6, 1-8, 1-9,  
 3-2  
 graphic design, 2-1, 2-2, 3-4, 6-4, G-3  
 graphic designer, 1-6, 2-2, 2-6, 2-8, 2-10, 2-11,  
 2-12, 2-13, 2-15, 2-16, 2-22, 3-3, 3-4, 3-20,  
 G-3  
 Graphical Kernel System standard (GKS), 3-6,  
 R-2  
 graphical resources, 2-27, 2-36  
 graphical style, xi, 3-1, 3-6, 3-17, 3-19, 3-20,  
 6-1, G-3  
 graphics package, 5-11  
 graphics parameters, 3-6  
 grid coordinate system, 5-13, 5-14, 5-21, 5-24  
 grid design, 5-1, 5-14, 5-15, 6-2, G-3  
 grid overlay, 5-22–5-23  
 GRID PACK, 5-4, 5-8  
 grid systems, 5-14, 5-46, 6-1  
 Griffin illustrator, xi, 3-3, 3-5, 3-6, 3-14, 3-18,  
 3-19, 5-10, 6-2  
 grouping data, 4-10, 4-13  
 Gruhn, Ann M., 2-26, R-4  
 gutter, 4-10, 4-12, 4-15, G-3

## H

hairline rule, 4-11, G-4  
 halftone, 2-6, 2-13, 2-19, G-4  
 halftone screens, 2-6, 2-13, 4-12  
 halign, in T<sub>E</sub>X, 4-19  
 Hammar, Michael, 2-34, R-4  
*Handbook of Computer-Aided Composition*,  
 1-6, 2-19  
 Helvetica typeface, 3-15, 4-13, 5-17  
 Hewitt, Carl, 5-9  
 hierarchical data structure, 2-34, 2-39–2-40,  
 3-7–3-11, 3-13, 3-16–3-18, 3-19, 5-12  
 hierarchical document models, 1-4, 5-13  
 high-quality illustrations, 1-1  
 high-speed typesetters, 2-21  
 horizontal alignment, *see* alignment,  
 horizontal  
 horizontal rule, 4-6, 4-11, 4-18, 4-19, 5-7,  
 5-14, 5-28, *see also* typographic rules  
 hot metal, 2-5, 2-19, G-4

house style, 2-2, 2-10, G-4  
 human vision, 4-7  
 Hurlburt, Allen, 5-15, R-4  
 Hypertext, 1-4, R-3  
 hyphenation, 2-16, 2-23, 2-30, 2-31, 4-7, 5-20,  
 G-4

## I

IBM 1130, 2-19  
 IBM 709, 2-19  
 IBM 7090, 4-3  
 IBM research, 2-34  
 ICCH3 (International Conference on  
 Computing in the Humanities, 1977), xii,  
 1-6  
 ideal, 2-25, 2-26, 2-27, 3-1, 3-5, 5-36, R-7  
 idiomatic graphics, 3-5  
 illustrations, xii, 1-8, 2-24, 2-25, 2-27, 2-29,  
 2-30, 2-35, 2-36, 2-39, 3-1-3-7, 3-9-3-12,  
 3-15, 4-6, 5-2  
 hierarchical structure, 3-10  
 high quality, 1-1, 1-8  
 line drawings, 1-1, 1-3, 1-4, 2-8, 2-12, 2-23,  
 2-26, 2-27, 2-34, 2-35, 2-37, 3-1, 3-2,  
 3-6, 3-18, 5-10  
 photographs, *see* illustrations, scanned  
 scanned, 1-1, 1-3, 1-4, 2-23, 2-30, 2-33,  
 2-35, 2-37, 3-1, 3-12, 3-19, 5-10  
 sketches, 2-11, 3-12-3-13  
 spelling checker within, 2-7  
 in tables, 4-6, 5-9, 5-10  
 three-dimensional, 3-1  
 illustrator, 2-8, G-4  
 bitmap, 3-5  
 Draw, 3-5  
 GOB, 3-5, 6-2  
 Griffin, *see* Griffin illustrator  
 Juno, *see* Juno illustrator  
 Markup, 3-5  
 Picc, 3-5  
 Picture, 3-5  
 Ilson, Richard, 2-34, R-4  
 image environment, 3-5, 3-19, R-3  
 imaging model, 5-30, *see also*  
 device-independent graphics package  
 imposition, 2-13, G-4  
 imprint page, 2-3, G-4  
 Incense graphical debugger, 5-9, 5-36  
 incremental formatter, 2-20, 2-24  
 indexer, 2-8, 2-11, G-4

indexing tools, 2-11  
 indices, 1-8, 2-3, 2-8  
 inputter, 2-12  
 integrated document composition, 1-9,  
 2-33-2-37, 6-1  
 interaction issues, 2-34, 6-4-6-5  
 interactive design tools, 2-27, 2-35, 4-4, 4-19,  
 6-1, 6-2, 6-4-6-5  
 interactive table formatter, 4-19, 5-33, 6-4  
 Interpress, 2-14, R-1  
 intersecting rules, 5-28-5-29  
 ISSCO, 3-5  
 italic typeface, 1-11, 4-10, 5-17, G-4

## J

jacket, 2-3, 2-10, 2-14, G-4  
 JACM style, 1-10  
 Janus, 2-1, 2-33, 2-34, 2-35, 2-39, 3-9, R-3  
 job docket, 2-2, 2-10  
 Johnson, S.C., 1-10, 2-26, R-4  
 journal editor, 2-8, G-4  
 Juno illustrator, 3-5, 3-20, 5-36, 6-2, R-5

## K

Kahn, Kenneth M., 5-9, R-4  
 Kernighan, Brian W., 2-5, 2-24, 2-25, 2-28,  
 3-5, 5-36, R-4  
 kerning, 2-13  
 keyboarder, 2-12, G-5  
 Khachiyani, L.G., 5-37, R-4  
 Kimura, Gary, 2-39, 6-2, R-4  
 Knott, Cargill Gilston, 4-13  
 Knuth, Donald E., 2-24, 2-30, 2-31, 2-32,  
 3-12, 3-15, 4-4, 4-19, R-4, R-5

## L

lack of integration, 2-7  
 Lamport, 2-31, 4-19, R-5  
 Lampson, Butler W., 2-37, R-5  
 language usage, 2-10, 2-11, 2-16  
 large tables, 4-10, 4-13-4-16, 5-43-5-44,  
 5-46  
 laser printers, 2-24, 2-38, 4-11, 4-12, 5-31  
 L<sup>A</sup>T<sub>E</sub>X, 2-31, 4-19, R-5  
 layout procedure, 3-10, 3-16, 5-3, 5-9, 5-11,  
 5-20, 5-22, 5-30, 5-44, 5-45  
 leaders, 4-7, 4-12, G-5

Leith, Robert, 2-13, R-5  
 Lesk, M.E., 1-9, 2-25, 2-26, 4-17, R-5  
 letter form design, 5-15, G-5  
 letterspacing, 2-16, G-5  
 Liang, Frank, 2-30, R-5  
 Library of Congress, 2-3  
 lifetime of illustrations, 3-4  
 line breaking algorithm, 2-30, 2-31, 5-19, G-5  
 line breaks, 4-7  
 line drawings, *see* illustrations, line drawings  
 line length hint, 5-20  
 line printer, 4-7, 4-16  
 line weight, 2-12  
 linear inequalities for table layout, 5-36  
 LINEAR INEQUALITY problem, 5-37, R-2  
 linear inequality solver, *see* constraint solver  
 linear optimization, 2-31  
 LINEAR PROGRAMMING problem, 5-37  
 lineup, 5-17  
 Linofilm phototypesetter, 4-3  
 Linotype, 4-11  
 Lipke, Daniel E., 2-36, 3-6, R-5  
 Liu, Joseph W-H., 1-7, 3-2, R-4  
 Lusignan, Serge, 1-6, R-5

## M

Macdonald, Nina H., 2-8, R-5  
 macro packages, 1-9, 1-10, 2-26, 2-28, 2-29, 2-35, G-5, *see also* troff, -me and troff, -ms  
 macro processor, 2-22, 2-27, G-5  
 macros, 2-24, 2-25, 2-27, 2-31  
 MacWrite, 2-38  
 manuscripts, 1-2, 2-8, G-5  
 mark, 5-17  
 marking up, 2-22  
 Markup illustrator, 3-5, R-6  
 master image, 2-13, G-5  
 mathematical composition, 2-5, 2-23, 2-24, 2-26, 2-28, 2-35, 2-36, G-5  
 mathematical notation, xii, 1-2, 1-4, 1-5, 1-6-1-7, 2-6, 2-33, 2-34, 3-2, 3-12, 4-1, 4-6  
   alignment of, 4-8, 5-17, 6-4  
   formatter, 2-25, 2-28, 2-30, 2-31  
   in tables, 4-6, 4-18, 4-19, 5-2, 5-17  
   two-dimensional nature, 5-11  
   typesetting, 2-12, 2-31  
 matrix notation, 2-28, 3-3, 6-4  
 mechanical artwork, 2-13, G-5  
 media, using different, 3-1, 3-4, 3-19, 6-2

METAFONT, 2-31, 3-15, R-5  
 metal type, 2-13  
 micros, 2-26  
 MIT Press, 2-2  
 MIT, 2-19, 2-34  
 Monier, Louis, 2-30, R-5  
 monospaced fonts, 1-6, 2-8, 2-24, 3-2, 4-7, 4-16, G-5  
 Monotype, 2-5, 4-11  
 moveable type, 1-2, G-5  
 Müller-Brockman, 5-15, R-5  
 Myers, Brad, 5-9, 5-36, R-5

## N

NBS, 2-19, 4-3  
 Nelson, Greg, 3-5, 3-20, 5-36, R-5  
 Nelson, Ted, 1-4, R-6  
 neqn, 2-26, *see also* eqn  
 Newman, William M., 2-3, 3-5, R-6  
 NEWSWHOLE, 5-16, 6-4, R-7  
 NLS, 1-4, 2-20, 2-21, 2-33, 2-37, 2-39, 2-40, 3-7, 3-9, R-3  
 nonhierarchical tables, 5-12, 5-13  
 nonlinear constraints, 5-36  
 NP-complete, 5-4-5-5  
 nroff, 2-24, *see also* troff

## O

object-oriented programming, 3-10, 4-19, 5-1, 5-9, 5-10, 5-21  
 office documents, 3-6  
 offset shadows, 3-17-3-18  
*One Book/Five Ways*, 2-2, 2-4, 2-5, 2-16, R-1  
 ornaments, 4-12, 5-29  
 Ossanna, Joseph F., 2-20, 2-24, R-6  
 Ousterhout, John K., 5-25, 5-33, R-6  
 overhead transparencies, 3-4  
 overlapping table entries, 5-28-5-29, 5-46

## P

page breaking, 2-21, 2-27, 2-31, G-5  
 page dummies, 2-6, G-5  
 page layout, 2-6, 2-21, 2-24, 2-27, 2-35, 4-1, 4-6, 4-12, 4-15, 5-1, 5-11, 6-2, 6-4, G-5  
 page makeup, 2-16, 4-4, 4-11, 4-12  
 page proofs, 2-11  
 page repros, 2-3, G-5

PAGE-1, 2-21, 2-28, R-6  
 pagination, 1-8, 2-23, 2-24, 2-28, 2-30, 2-31  
 paint program, 2-6, 3-18, R-2  
 painter's algorithm, 5-29, G-5  
 panel, of table, 4-5  
 paragraphs within tables, 5-19  
 parallel processes, 2-6  
 PARTITION PROBLEM, 5-4—5-5  
 PASCAL, xii, 1-5, 2-35, 3-2, R-3  
 paste-up artist, 2-13  
 paste-up, 2-21, 3-19  
 Paxton, William, xi, xii, 2-11, R-7  
 Peck, Russell A., 1-5, R-6  
 pen plotters, 3-2  
 penalty copy, 2-5, G-5  
 Phillips, Arthur H., 2-19, 4-1, 4-2—4-3, 4-4, 4-14, R-6  
 photographs, 2-13, 5-4, 5-10, *see also*  
   illustrations, scanned  
 Photon 560, 4-3  
 Photon 737, 2-19, 2-21  
 Photon Econosetter, xii, 2-21  
 phototypesetters, 2-5, 4-11, 4-12, G-5  
 pic, 2-25, 2-26, 2-27, 3-1, 3-5, 4-18, 5-36  
 Picc, 3-5, R-7  
 picture description languages, 3-5  
 Picture illustration system, 3-5, R-2  
 Pierson, John, 2-21, 2-28, R-6  
 Plass, Michael F., xii, 2-30, 2-31, 4-5, 4-14, R-6  
 polynomials, alignment of, 4-8  
 PostScript, 2-14, R-1  
 pregnant pause syndrome, 1-8  
 preprocessors, 2-25, 2-26, 2-28, 2-30, *see also*  
   troff  
 Press print file format, 2-37  
*Principles of Interactive Computer Graphics*,  
   2-3, R-6  
 printing plates, 1-2, 2-13, G-6  
 printing systems, 1-2  
 process camera operator, 2-13  
 production difficulties, 1-8  
 production editor, 2-2, 2-4, 2-8, 2-9, G-6  
 PROFF, 2-19, 2-21  
 proofreading, 1-5, 1-6  
 property sheet, 3-6, 6-2  
 prototype  
   TABLE, *see* TABLE interactive formatting  
   prototype  
   table, *see* table, formatting prototype  
   TiogaArtwork, *see* TiogaArtwork  
 punctuations, 2-16

## R

RANDOM PACK, 5-4—5-5, 5-6  
 raster images, *see* illustrations, scanned  
 RCA Videocomp, 2-21  
 readability, 4-4, 4-7, 4-13, 4-15, 5-5, 6-3  
 recomputing constraint system, 5-36  
 refer, 2-25  
 referees, 2-9  
 Reid, Brian, 1-9, 2-22, 2-29, R-6  
 rendering procedure, 3-10, 3-16, 5-3, 5-9, 5-11, 5-28, 5-30—5-31  
 reproduction pages, 2-3, G-6  
 resolution  
   device, 2-18, 2-38, 5-11, 5-31  
   typesetters, 4-7  
 Reston Publishing Co., 3-3, 3-20  
 reviewers, 2-9  
 Ritchie, Bob, xi  
 Ritchie, Dennis, 2-24, R-6  
 Rosenthal, David, 2-27, R-6  
 row depth, 4-4  
 row hierarchy, 5-12—5-13  
 row structure, 4-4, 5-2  
 row style, 4-10  
 rules, *see* typographic rules  
 running head, 1-7, 2-13, 2-21, 6-2, G-6  
 RUNNOFF, 2-20, 2-21, 4-17, R-6

## S

Saltzer, J., 2-20, R-6  
 sample pages, 1-10, 2-6, 2-16, G-6  
 scanned images, *see* illustrations, scanned  
 scholarly publishers, 4-11  
 Scientific American illustrations, 1-10, R-2  
 Scribe, 1-9, 2-1, 2-20, 2-22, 2-24, 2-28, 2-29, 2-34, 2-39, 4-16, R-1, R-6  
 selection hierarchy, 5-2, 5-12—5-13, 5-24, 5-46  
 selection mechanism, 4-20, 5-1, 5-2  
 separations, 2-14  
 Seybold, John W., 2-19, 2-21, 2-36, 2-38, 4-7, R-6  
 shadows, 3-17—3-18, 6-2  
 Shand, Mark, 5-25, 5-32, R-6  
 Shaw, Alan C., 2-39, R-4  
 shift codes, 2-19  
 Shiloach, Yossi, 5-37, R-2  
 SIGGRAPH'83, xi, 3-1  
 signatures, 2-13, 2-14, G-6

Simplex method, 5-37  
 Sketchpad, 5-35, R-7  
 slack variables, 5-36  
 Smith, David Canfield, 2-36, R-6  
 spanning heads, 4-5, 4-7, 4-9, 4-10, 4-13, 4-16,  
   4-17, 5-2, 5-8, 5-12, 5-14, 5-21, 5-24  
 sparse matrices, 1-7, 3-2  
 special settings, 2-16  
 specification sheet, 2-10  
 spelling checker tools, 1-7, 2-7, 2-9, 2-11,  
   2-25, 2-40  
 spelling, 2-10, 2-17  
   Canadian *versus* American, 2-17  
 spreadsheets, 1-7, 4-3, 4-15, 4-16, 6-3  
 Sproull, R.F., 2-3, R-6  
 square table entries, 5-15, 5-43, 6-3  
 statistical tables, 4-14  
 Stevens, Mary Elizabeth, 4-3, R-7  
 Stone, Maureen, xi, 3-1, 3-5, 3-17, 5-10, R-2  
 Strachey, C., 2-28, R-6  
 stripper, 2-13  
 stub heads, 4-14, 4-16, G-6  
 STUB PACK PROBLEM, 5-6—5-8  
 style, 1-2, 1-9, 2-16, 2-17, 2-24, 2-25, 2-26,  
   2-27, 2-9, 2-35, 2-36, 3-5, 3-9, 3-20, G-6  
   analysis aids, 2-11  
   book, 1-6, 2-16, *see also* style, manual  
   control, 2-28  
   consistency, 2-12, 3-1, 3-4, 3-19  
   definition of, , 2-22  
   dictionary of style rules, 3-8, 3-18, 3-20  
   document, *see* document style  
   graphical, *see* graphical style  
   guidelines, 3-4  
   house, 2-2, 2-10  
   lifetime, 3-4  
   machinery, 5-3  
   manual, 1-10, 6-5  
   publisher's, 2-16  
   rules, *see* style rule  
   sheet, 2-10, 2-12, 2-22, 6-4  
   for tables, 4-10, 4-18, 5-16—5-19  
 style rule, 3-12, 4-6, 5-3, 5-16  
   search path for applying to table, 5-16  
 subscripts, 2-26  
 superscripts, 2-26, 4-13  
 Sutherland, I.E., 5-35, R-6  
 symbolic algebra, 1-7  
 syntax checker for eqn, 2-23  
 systems of equations, alignment of, 4-8

## T

tab stops, 4-16—4-17  
 table, 4-1, G-6  
   alignment, 5-2  
   box head, 4-5, 4-6  
   broadside, 4-15  
   entry, *see* table entry  
   formatting of, 2-24, 2-25, 2-27, 2-28, 2-30,  
     2-33, 2-34, 2-35, 2-36, 2-39, 3-12, 3-19,  
     4-3, 4-4, 4-17—4-18, 4-19, 6-1  
   formatting prototype, 5-10, 5-13, 5-45, 6-1,  
     6-2  
   geometry, 4-4, 5-46  
   layout, 4-4, 5-1, 6-4  
   manuscript preparation, 2-5  
   nonhierarchical, 5-12, 5-13  
   object representation, 5-22—5-23, 5-26,  
     5-27  
   panel, 4-5, 4-6  
   reduced typesize, 4-13  
   spanning heads, 4-5, 4-6  
   stub, 4-5, 4-6  
   style, 4-18, 5-2  
   topology, 4-4, 5-13, 5-14, 5-21, 5-25, 5-33  
   upright, 4-15  
   vertical rules in, 2-5  
 table entry, 4-6, 4-18, 5-33, G-6  
   data structure, 5-28  
   overlapping, 5-28—5-29  
 TABLE interactive formatting prototype, 2-27,  
   4-19—4-20, R-2  
 tableau of constraints, 5-36  
 tables of contents, 1-3, 1-7  
 tables of numeric data, 1-7, 4-2, 4-3  
 TABPRINT, 4-3—4-4  
 tabular composition, 1-7—1-8, 2-23, 2-31, G-7  
 tabular information, 1-1, 1-2, 1-4  
 tag definition, 2-22  
 tags, formatting, *see* formatting tags  
`tb1`, 2-25—2-27, 2-33, 4-17—4-18, 4-19, 4-20,  
   5-8, 5-12, 5-19, 5-20, R-5  
 Teitelman, Warren, 1-2, 2-37, 3-1, 3-7, R-7  
 TELE-A-GRAF, 3-5, R-1  
 terminals, 1-4  
 tessellations, *see* corner stitching, tessellations  
 TEX, 1-6, 2-1, 2-21, 2-24, 2-31, 2-34, 2-35,  
   2-40, 4-19, 5-9, R-5  
*TeXbook*, *The*, 2-30, 2-32, 2-33, 2-34, 3-12,  
   R-5  
 text composition, 2-16  
 text editor program, 2-8, G-7

- Thinglab, 5-35—5-36, R-2  
 Thomas, Elaine L., 3-6, R-7  
 three-dimensional images, 3-1  
 Tilbrook, David M., 5-15, 6-4, R-7  
 Times Roman typeface, 2-22, 2-38, 3-15, 5-17  
 Tioga, xii, 1-2, 2-37, 2-40, 3-8, 3-9, 5-46, 6-1  
   document model, 1-2, 3-7—3-9, 5-3, 5-20, 5-45  
   node properties, 3-9  
   style machinery, 1-2, 3-7, 3-15, 5-3, 5-45  
   text editor, 5-21  
 TiogaArtwork, 3-1, 3-6—3-7, 3-10, 3-13, 3-10, 3-13, 3-14, 3-15, 3-17, 3-18, 3-19, 3-20  
 topology, 4-4, 4-19, 5-3, 5-21  
 totals, 4-10, 4-14—4-15  
 touch-ups, 1-8  
 traditional graphic arts process, 2-1, 2-2—14  
 trajectories, 3-12  
 transfer lettering, 3-4, 4-12  
 transformations, 3-10, 3-12, 3-13, 3-14, 3-16, 3-17, 3-18, 3-19, 3-20  
 transposing tables, 4-13—4-14, 5-2  
 trapezoidal rule illustration, 3-12—3-13, 3-18—3-20  
 traps, 2-27, 2-28, *see also* diversions  
 troff, 1-9, 2-1, 2-20, 2-23, 2-24—2-29, 2-30, 2-35, 2-39, 3-5, 4-17, 4-18  
   bibliographic reference formatter, *see* refer  
   device-independent, 2-24, 2-28  
   ditroff, 2-24  
   equation formatter, *see* eqn, neqn  
   illustrations, *see* pic, ideal  
   limited registers, 2-27, 4-18, 4-20  
   macro packages, 1-9, 2-26  
   -me macro package, 2-26, R-1  
   -ms macro package, 1-9, 2-26, R-5  
   nroff, 2-24  
   performance issues, 4-20  
   porting to other devices, 2-24, 2-29  
   preprocessor interaction, 4-18, 5-9  
   table formatter, *see* tbl  
 Turing lecture, 2-24, R-6  
 two-dimensional layout, 4-1, 4-5—4-6, 4-7, 5-1, 5-2, 5-3  
 TWO-DIMENSIONAL PACKING problem, 5-4, 5-6, R-2  
 two-page spread, 4-15  
 typeface, G-7  
   Baskerville, 2-18, 2-38  
   bold typeface, 3-16, 4-10, 5-17  
   compressed, 4-13  
   typeface (continued)  
     Garamond, 2-18, 2-38  
     Helvetica, 3-15, 4-13, 5-17  
     italic, 1-11, 3-16, 4-10, 5-17  
     Times Roman, 2-22, 2-38, 3-15, 5-17  
 TYPESET, 1-5, 2-20, R-2  
 typewriter, 2-8, 4-7, 4-16  
 typist, 2-8  
 typographic grids, 5-13, 5-15  
 typographic rules, 4-6, 4-7, 4-10—4-11, 4-18, 5-2, 5-14, 5-21, 5-24, 5-25, 5-28, 5-30, 5-46  
 typography, 1-9, 4-2, G-7  
 Tyxset, 2-32, R-1
- U*
- undo constraints, 5-36  
 undo, 6-2  
 University of Chicago Press, 2-2, 2-5, 4-11  
 University of North Carolina Press, 2-2  
 University of Texas Press, 2-2  
 University of Toronto Press, 2-2, 2-16  
 University of Waterloo, xi, 2-19, 2-21  
 UNIX, 2-24, 2-25  
   filter/pipe model, 2-25  
   stream model, 5-12  
   troff, *see* troff  
   writer's workbench, 2-8  
 Updike, Daniel Berkeley, 1-9, R-7  
 user interface, 2-39, 6-1
- V*
- van Leunen, Mary-Claire, 1-6, 2-17, 2-40, R-7  
 van Wyk, Christopher J., 2-25, 3-6, 5-36, R-7  
 variable width characters, 2-24  
 vertical alignment, *see* alignment, vertical  
 vertical rule, 2-5, 2-16, 4-6, 4-10, 4-12, 4-18, 4-19, 5-14, 5-24, 5-28, G-7, *see also* typographic rules  
 views, 2-10, 2-18, 2-21, 2-40, 6-2, R-3  
 VLSI layout, 5-25
- W*
- Warnock, John, 3-1, 3-7, 5-30, R-7  
 Waterloo Computer Typography, *see* WATTYPE  
 WATTYPE  
 WATTYPE-S, xii, 1-5, R-3

WAITYPE (Waterloo Computer Typography),  
   xii, xiii, 1-5, 1-7, 1-8, 2-28  
 waxing machine, 2-13  
 weight of rule, 4-11  
 'what you see is almost what you get', 2-38  
 'what you see is what you get', *see* WYSIWYG  
 White, Alex R., xii, 3-5, R-7  
 whitespace, 2-22, 2-23, 2-30, 4-7, 4-8, 4-10,  
   4-13, 5-4, 5-8, 5-19, 5-20, 5-40, 6-3, G-7  
 wide tables, 4-15–4-16  
 widows, 2-23  
 Williamson, Hugh, 1-9, 2-5, 3-2, 3-4, 4-11,  
   4-13, 4-15, 5-15, R-7  
 Winograd, Terry, 2-11, R-7  
 Witten, Ian H., 2-27, R-7  
 wordspaces, 5-20, G-7  
 writer's workbench, 2-8, 2-29, R-5  
 writing tools, 2-8  
 Wyatt, Doug, xii, 3-1, 3-7, 5-30, R-7  
 WYSIWYG, 1-7, 1-9, 2-20, 2-33, 2-34, 2-36,  
   2-37, 2-38, 2-39, 3-10, 3-18, 4-19, 5-1, 5-11,  
   5-16, 5-21, 5-31, 5-46, G-7

## X

Xanadu, 1-4, R-6  
 Xenix, 2-32  
 xerography inventor, 5-10  
 Xerox, 2-14  
 Xerox PARC, xi, 1-2, 2-4, 2-33, 2-37, 3-1, 3-5,  
   3-9, 3-19, 5-10, 6-1  
 Xerox Star, 2-1, 2-33, 2-36, 2-37, 2-38, 3-2,  
   3-6, 3-7, 3-20, 6-2  
   graphics, 2-36, 3-6, R-5  
   interface, 2-39

## Y

Yorktown formatting language (YFL), 2-26,  
   R-4

## Z

Zabula-Salelles, Ignatio Andres, 3-5, R-7  
 Zeisel, Hans, 4-14, R-7



# Colophon

---

This thesis was formatted and printed at Xerox PARC. Multiple original copies were printed on a Xerox XP-12 laser printer, which was modified to have a resolution of 384 dots per inch and was configured as the Press printer named RockNRoll on the Xerox Research Internet. The fonts were created at PARC as 384-dot-per-inch bitmaps from spline outlines. The type families include TimesRoman, Helvetica, Xerox Book, Cream, Old English, Math, and Hippo (Greek) in various weights (regular, bold, italic, and bold-italic) and in various sizes (6, 8, 9, 10, 12, 14, 18, and 36 points).

The manuscript of the thesis exists as many document files, one per chapter plus ancillary illustration files. The manuscript was edited with the Tioga document composition system running in the Cedar programming environment on a personal Dorado high-performance workstation named Shangrila. Formatting was accomplished with the Tioga typesetter and prototype implementations of TiogaArtwork (illustrations) and the TableTool (tables). Line drawing illustrations were created with the Griffin illustrator and included either as TiogaArtwork illustrations or a preformatted Press files. Scanned images were created by a server on the Xerox Research Internet using an Eikonix scanner and were incorporated into the formatted document with TiogaArtwork.