CONFORM: **A Context for Electronic Layout -**
Relating Content to Form
for Authors and Designers

Susan Wascher
BFA Western Michigan University
1977

Submitted to the Department of Architecture
in partial fulfillment of the requirements
of the degree
Master of Science in Visual Studies
at the **Massachusetts Institute of Technology**

June 1985

signature of author

Susan Wascher
Department of Architecture
May 15, 1985

certified by

Muriel Cooper
Associate Professor of Visual Studies
Thesis Supervisor

accepted by

Nicholas Negroponte
Chairman, Departmental Committee on
Graduate Students

CONFORM: **A Context for Electronic Layout** –
Relating Content to Form
for Authors and Designers

Susan Wascher

Submitted to the Department of Architecture
in partial fulfillment of the requirements
of the degree

Master of Science in Visual Studies
at the **Massachusetts Institute of Technology**

Abstract

CONFORM is a software package concerned with the spatial and temporal layout of documents that integrate text and graphics. Developing a method for chunking and organizing the information contained in such publications is the first step in creating a system that will assist a designer with his or her task. The aim is to use the content analysis of the document as a means for defining its final form. It is based on the proposition that the visual form of the document should be in agreement with its content.

Content is defined in terms of a linguistic hierarchical structure with links to specific images to be included in the document. Form is described through a typographic style sheet and page regions. Content helps define Form by using the hierarchical structure to generate the typographic parameters, and to set the text in the appropriate page regions.

Thesis Supervisor: Muriel Cooper
Title: Associate Professor of Visual Studies

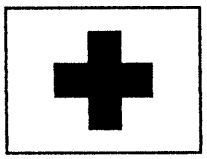
Contents

Introduction

The need to organize information and present it in a form that encourages the easy, efficient, and thorough assimilation of information is more important today than ever before. Most of us, 60% of the U.S. workforce, are involved in creating, processing, or distributing information rather than producing goods. Information is being generated at such a rate that it is difficult for any one person to synthesize and assimilate. If it is hard to access, or difficult to read, the lack of information, whether incomplete or misunderstood, limits our understanding and hinders progress. For instance, between 6,000 and 7,000 scientific articles are written each day. [NAISBITT] This is more than the scientific community can keep up with, and often results in duplication of efforts.

The design of information is an important and nontrivial task. In addition to understanding the objective of the message and its intended receiver, it is important to have some knowledge of communication theory, visual perception, and cognitive processes. These disciplines have much

to contribute, not only to the understanding of visual communication, but to the process of design as well. Communication theorists provide us with the means to quantify information, understand the constraints of a communication system, and evaluate the end result. Greater effectiveness in organizing and synthesizing information can be achieved through understanding our own cognitive processes. And, since visual communication relies on our ability to see and recognize what we see, theories of visual perception can aid us in creating and arranging effective visual representations.

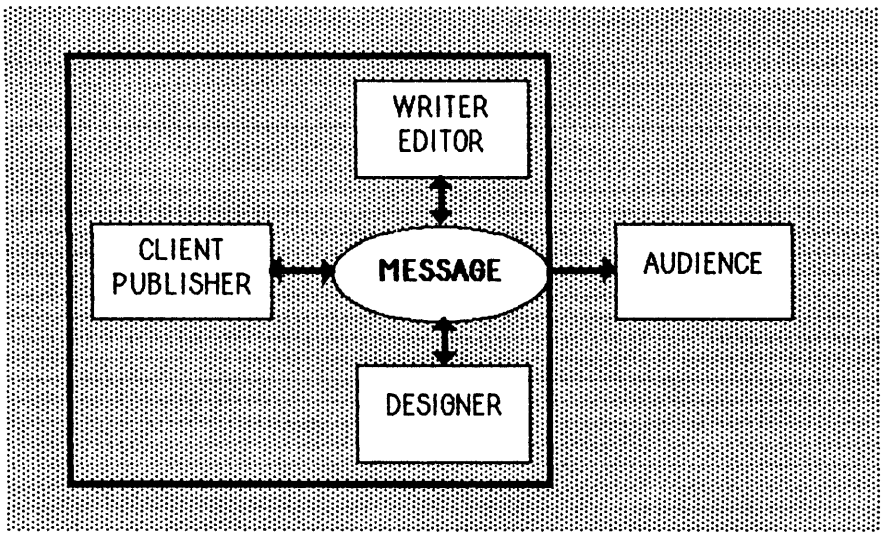


Red Cross

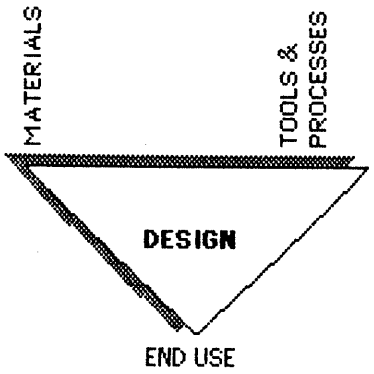
Red Cross

Visual communication employs both visual and verbal symbols to encode and transmit a message. These symbols take the general form of text and image which are embodied in various typographic and graphic styles. Using the linguistic structure of the message to determine an appropriate style that will aid comprehension is the subject of this thesis. The general project domain consists of the integration of text and graphics in an electronic publishing environment.

The following diagram depicts the process of molding a message for the end receiver. A CLIENT (marketing executive, company president, government agency, etc.) or PUBLISHER identifies the audience and purpose of the message, the WRITER/EDITOR creates the text, and the DESIGNER gives it visual form. These shouldn't be distinct functions, but integrated to produce an effective message that has meaning for the intended AUDIENCE. The task of the information designer is to relate the visual and verbal material so that the content of the message is supported, whether it be in print or electronic form.



The task of creating an effective message involves forming it verbally and visually. The functions may be performed by a combination of individuals or by a single person.



Triangle of Design Forces

J Bronowski described design as taking place within a triangle of forces: the tools and processes by which an object is made, the materials of which it is made, and the end use.

Ralph Caplan

The process of designing an effective communication piece takes many forms and approaches. If you were to ask ten different designers to define their design process, you would probably receive ten different answers. Some attempts at description include:

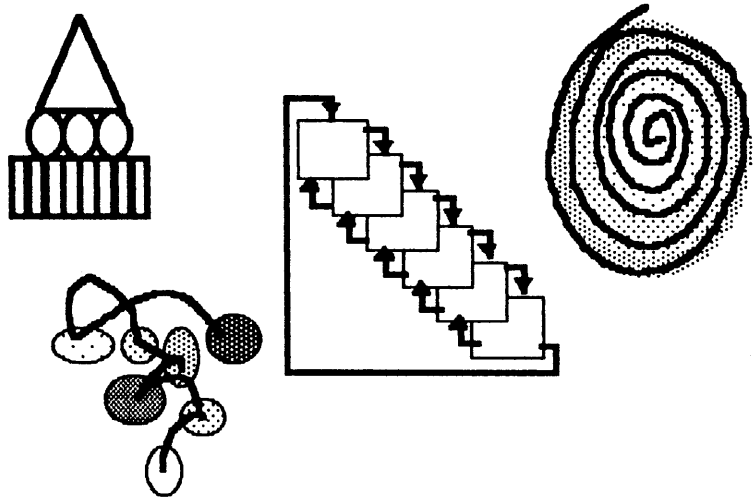
"...seizing on a purpose, defining the situation or problem, identifying constraints and organizing materials, people and events in a way that can be modelled and visualized in advance." [CAPLAN]

"...processes of judging, deciding, choosing, and creating." [SIMON]

"...the expression "design process" we might say embraces on the one hand the ensemble of qualities and technics necessary in planning, and on the other the determination of their actual application." [OLIVETTI]

"...process of input, examination, and decision-making..." [SCITEX]

"Design is choice ... What is to be sought in designs for the display of information is the clear portrayal of complexity. Not the complication of the simple; rather the task of the designer is to give visual access to the subtle and the difficult - that is, the revelation of the complex." [TUFTE]



Design theories should suppose that we all have a bit of serialist and wholist in us, varying at different times, over different subjects, in different searchings.

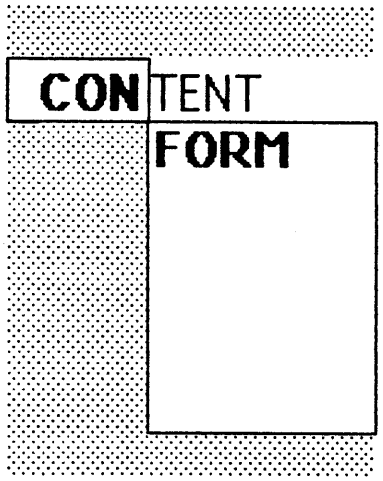
Nicholas Negroponte

Whether the process is top-down, bottom-up, step-wise refinement, linear, or circular, it generally consists of defining the situation or problem; identifying constraints, gathering and organizing materials; synthesizing the information; visualizing and producing a solution; and evaluating the result. It can be characterized as problem-solving, goal-making, or "search for sufficient, not necessary actions for attaining goals." [SIMON]

Due to the great complexity and quantity of information being produced, computer applications are necessary that will aid the design process in analyzing, synthesizing, and visualizing information, plus provide assistance to the user in making appropriate style choices.

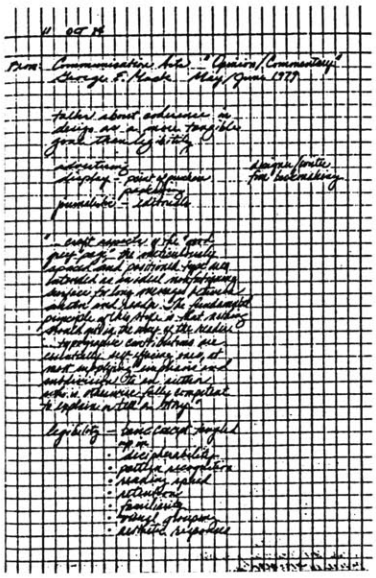
As part of this thesis, and to meet these needs, a computer application named CONFORM was developed as one approach to designing a system that creates relationships between a document's content and the visual components that give it form. CONFORM provides a means for the author or designer to define the text's content in terms of its linguistic hierarchical structure (title, headings, subheads, and body), and create links to images. By supplying typographic rules and default parameters, a typographic style sheet is generated based on this hierarchical structure. In order to provide the user with the opportunity to preview the document and evaluate its design, a page represented as regions with levels of the structure associated with them is automatically formatted in the typographic parameters specified in the style sheet.

CONFORM aims to provide a means of organizing and integrating the various pieces of information that need to be communicated, and to create a context from which a format can be derived.



The English word "conform" is derived from the Latin word "conformare", which means to have the same form, or shape after. Today it means to act or be in agreement with something, such as "You must conform to today's social standards." CONFORM as the title of this thesis supports both meanings; the format or form of a document should be shaped after its content, and content and form must be in agreement to support understanding. In other words, form follows content.

The word "content" has many meanings which include: all that a thing contains or deals with, subject matter, meaning or significance, holding capacity or size, and quantity of a specified part. Content of information refers to the meaning of the message, and what is contained in it, such as subject matter, tone or attitude, major and minor themes, or levels of detail.

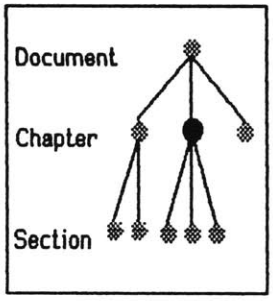


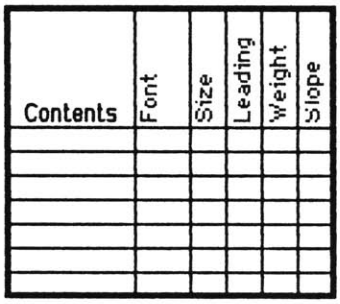
Of the numerous definitions of form, the one that pertains to this discussion states that form is a style, manner, and procedure, as opposed to content. [DOUBLEDAY] There are two basic message forms, the one the originator creates the message in (gives form to his or her idea), and the one it is executed in for an audience. Meaning is graphically expressed every time we take pen to paper or type at a keyboard, and may be enhanced through typography. Moving from one form to the other on the basis of content is supported by CONFORM.

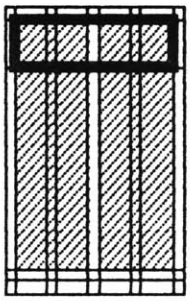
One approach to information design is to perform a content analysis of the manuscript to determine its **objective**, whether it is to persuade, inform or question; its **significance**, importance, consequence, or value; and its **complexity**, the number and distribution of major and minor points or levels of detail. Of these three types of content analysis, complexity is best suited to computer assistance since it requires parsing a manuscript into levels of information, relating the levels to one another, and calculating their number and distribution.

This analysis is used to determine the formal arrangement and visualization of the message. Levels of emphasis can be depicted through typographic parameters and spatial arrangement. Themes and meanings may also be amplified through visual support.

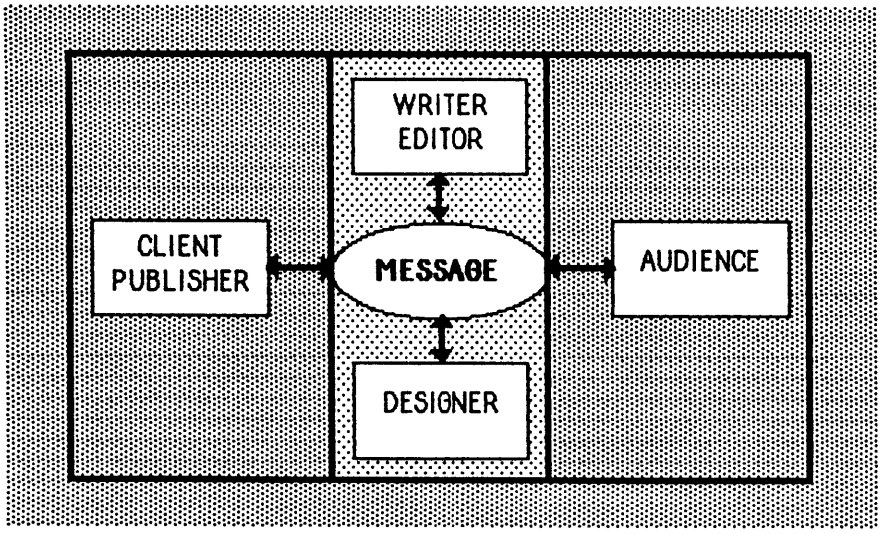
CONFORM supports the quantitative analysis of the manuscript's content by the designer or author into a hierarchical structure of linguistic categories. Form is described through a combination of representations for page constraints (grid and regions) and typographic parameters (style sheet). Relating images to text or identifying images that need to be created is also supported.







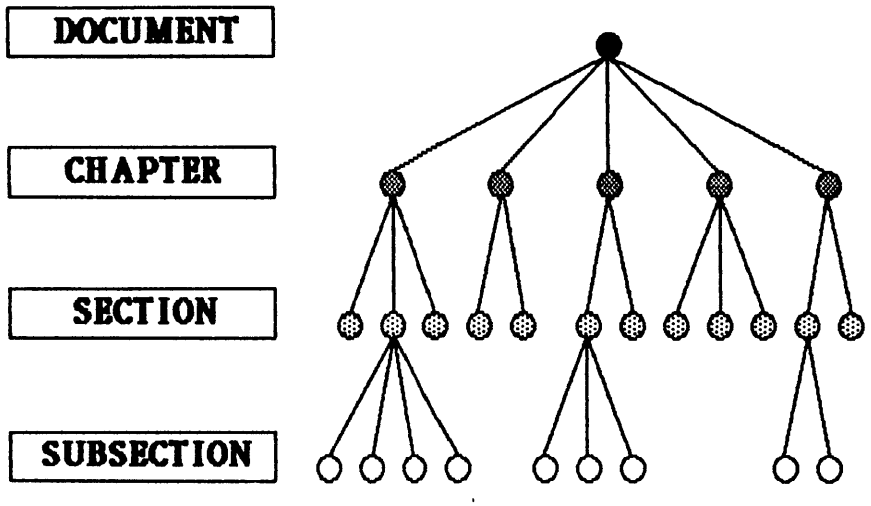
Creating separate representations for content and form allows greater flexibility in designing a document. Supplying rules and default values for generating the form parameters based on the content structure, gives the author of a manuscript the ability to create an effective message without having to learn the esthetics of typography. In fact, even the reader of the document may view it in a form that meets their own requirements, such as larger type for the seeing impaired. With this added ability, the earlier diagram that illustrated the actors or functions that influence the form of a message needs to be revised to include the AUDIENCE as an active participant.



Modern architecture's familiar slogan "form follows function", attributed to the American architect Louis Sullivan, is a slogan for an industrialized society wrestling with the complexity and fragmentation brought about by the machine, trying to integrate art and industry. Though our society has shifted from an industrial base to an information base, the concept of form follows function, or stated here as form follows content, is still relevant. More than a slogan, it is based on theories of perception and cognitive psychology. In making a distinction between the words "shape" and "form", Rudolf Arnheim in his book Art and Visual Perception describes shape as "the spatial aspects of appearance", and goes on to say, "But no visual pattern is only itself. It always represents something beyond its own individual existence - which is like saying that all shape is the form of some content."

To provide background information, and place this work in perspective, the structure of information and existing electronic publishing systems for the integration of text and image will be discussed. The main body of this document describes the project and its design and implementation. Illustration of an example session using the software created is also included. The conclusion discusses the success of this approach for formatting documents, plus considerations for broader applications and future work. Software documentation is in an appendix for those interested in building on this work. All references made in the body of this document are listed in the bibliography.

Information Structure



Information Structure

The structure of information is discussed in a variety of literature from the designer's perspective to the information theorist's point of view. Karl Gerstner in his book A Compendium for Literates cites the work of information theorist Kurd Alsleben:

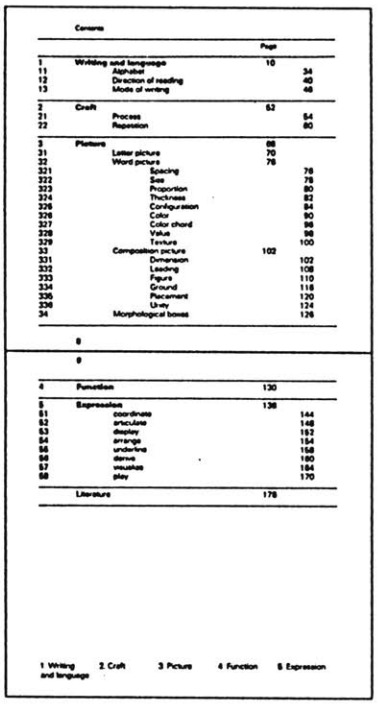


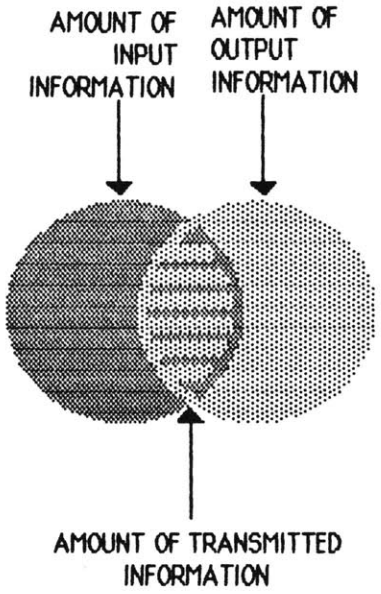
Table of Contents to
A Compendium for Literates

Alsleben: "As a rule the message of a text can be seized the more readily the easier it is to see how each part of the text relates to the context of the whole. That is, it pays to show the structure of the text. Certain typographic resources allow it to be displayed with less trouble than composition and typeface can do alone."

"One task of typography is exemplified by a table of contents: imagine a table of contents as a continuously set text, without paragraphs or indentations; it would have to be read and reread before any idea of structure came through. Imagine, too, what an effort it would cost to rectify this limitation with words of explanation. This explanatory function is taken over by typographic means."

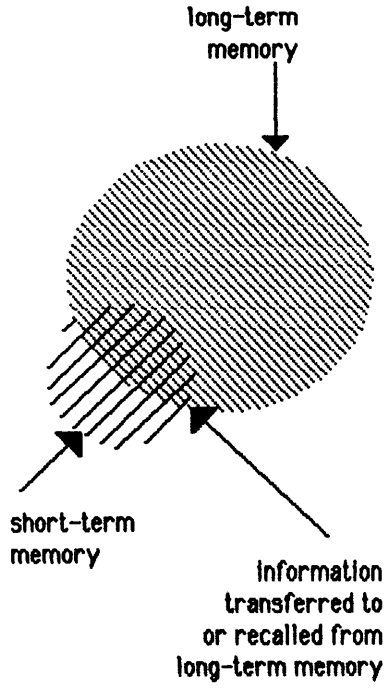
"The principal difficulty for a text lies in playing down the inevitable successivity of what it says. The inevitability of this successivity does not originate in particular from the possible ways a piece of printing can be produced but rather from the fact that the mind can take in only a small amount of information, about 160 bits at a time. To overcome this difficulty it reduces the supply of information by segmenting it into lumps of an acceptable size."

In 1956, psychologist George Miller introduced the word "chunk" to describe units of memory. Using concepts and measures from Claude Shannon's work in information theory, Miller quantitatively determined the limitations on our capacity to receive, transmit, and remember information. He concluded that our absolute judgment is limited by the amount of information, and our immediate memory is limited by the number of items.



Graphic derived from Miller's description of experiments on absolute judgment. The channel capacity (transmitted information) is the greatest amount of information a human observer can give us about a stimulus.

Absolute judgment refers to the accuracy in which we are able to identify or recognize information. The amount of information is measured in bits; one bit represents the amount of information needed to make a decision between two equally likely alternatives, two bits for 4 alternatives, three bits for 8, and so on. For instance, 1 bit is the amount of information needed to determine whether a man is less than or greater than 6 feet tall. However, our everyday decisions consist of making crude judgments of several things simultaneously. Miller states that the limit to our accuracy in identifying a unidimensional stimulus is about 7 bits, but increasing the dimensions along which something may vary, and requiring relative instead of accurate judgments, increases this limit to at least 150. This corresponds to Alsleben's figure of 160 bits of information.



Immediate memory, also referred to as short-term memory, contains information of which we are immediately aware, either from an outside stimulus, or recalled from long-term memory. The amount of information recalled from either source is a fixed number of items or chunks, somewhere around seven, but what constitutes a chunk is unclear. Chunks of information are made up of some number of bits; the number varies depending on how we organize and group the information. Miller states that, ". . . we must recognize the importance of grouping or organizing the input sequence into units or chunks. Since the memory span is a fixed number of chunks, we can increase the number of bits of information that it contains simply by building larger and larger chunks, each chunk containing more information than before." [MILLER]

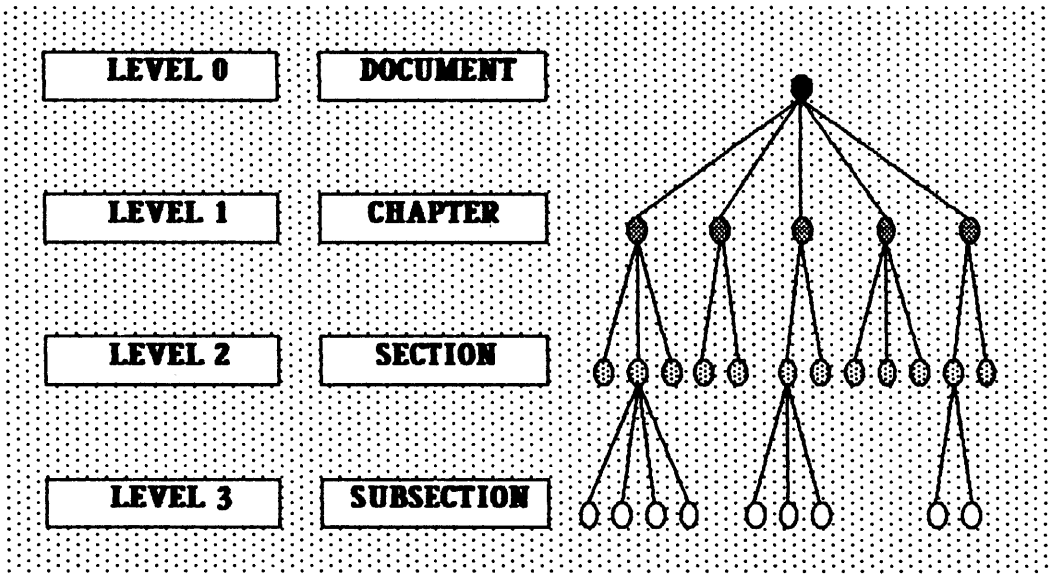
Audience comprehension can be aided by careful chunking of a message into idea segments which are then given visual form. Form can express and amplify the interrelationships of the segments, character of the message originator, and significance of the message.

John Anderson describes the various factors entailed in comprehension as:

"First, we use our knowledge of the general syntactic and semantic patterns of the language to translate from words to meaning. Second, our general knowledge of the supposition-assertion distinction helps us to relate new information to old. Relevant also are general cognitive skills as reasoning and problem solving plus general world knowledge. Finally, our knowledge of typical text structure helps us to perceive the relations between relatively large portions of linguistic input." [ANDERSON]

Text Structure

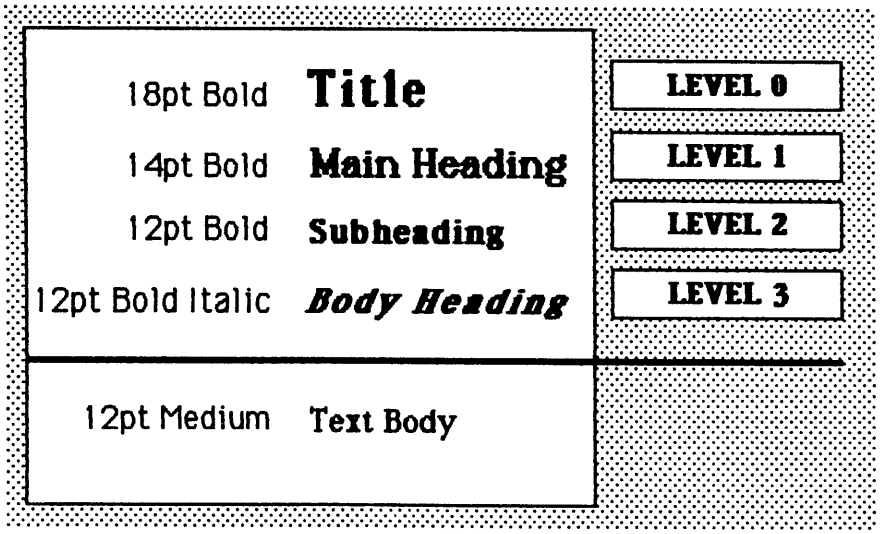
We tend to remember or recall things in relation to other things; something triggers an idea, and that triggers another. Organizing a document into chapters, sections, subsections, and so on, creates a hierarchical structure of idea segments that not only aids recall, but understanding as well through recognition of the relationship of the segments.



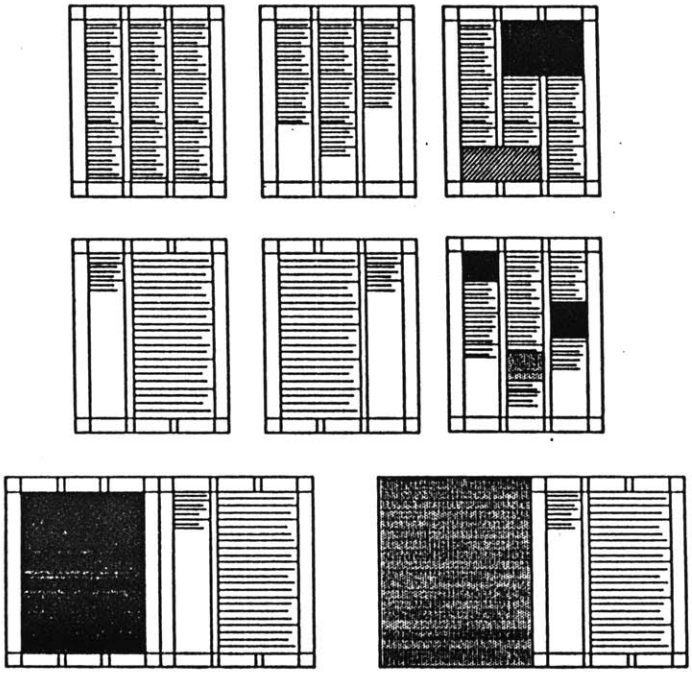
"Linguistic units larger than sentences, such as paragraphs and texts, are structured hierarchically according to certrain relations. Information higher in a text structure tends to be better recalled than that lower in the structure. Comprehension of a text depends critically on the perceivers ability to identify the higher order structures that organize it." [ANDERSON]

Visualizing these higher order structures is accomplished through spatial arrangement, temporal sequence, and typographic emphasis. "By carefully arranging type areas, spacing, size, and 'color', the typographer is able to impart to a printed page a quality which helps to dramatize the contents." [RAND]

Typographic emphasis is achieved through visual contrast of size, shape, weight, spacing, color, or position. For example, the following illustration depicts four levels of text headings plus the text body. Changing the point size to correspond to the levels shows how visual emphasis is created and identifies the text structure.



Visually organizing and ordering the text is another means of identifying its structure. The division of space on a page into specific text areas creates a physical structure that supports the organization of the information. It can also indicate how images are integrated into the structure. Flow or rhythm of the information is created by breaking it over pages at idea segments, and pacing the amount of information that is presented to the reader. These same concepts can be applied to frames of information displayed on a video monitor with the screen taking the place of the page.

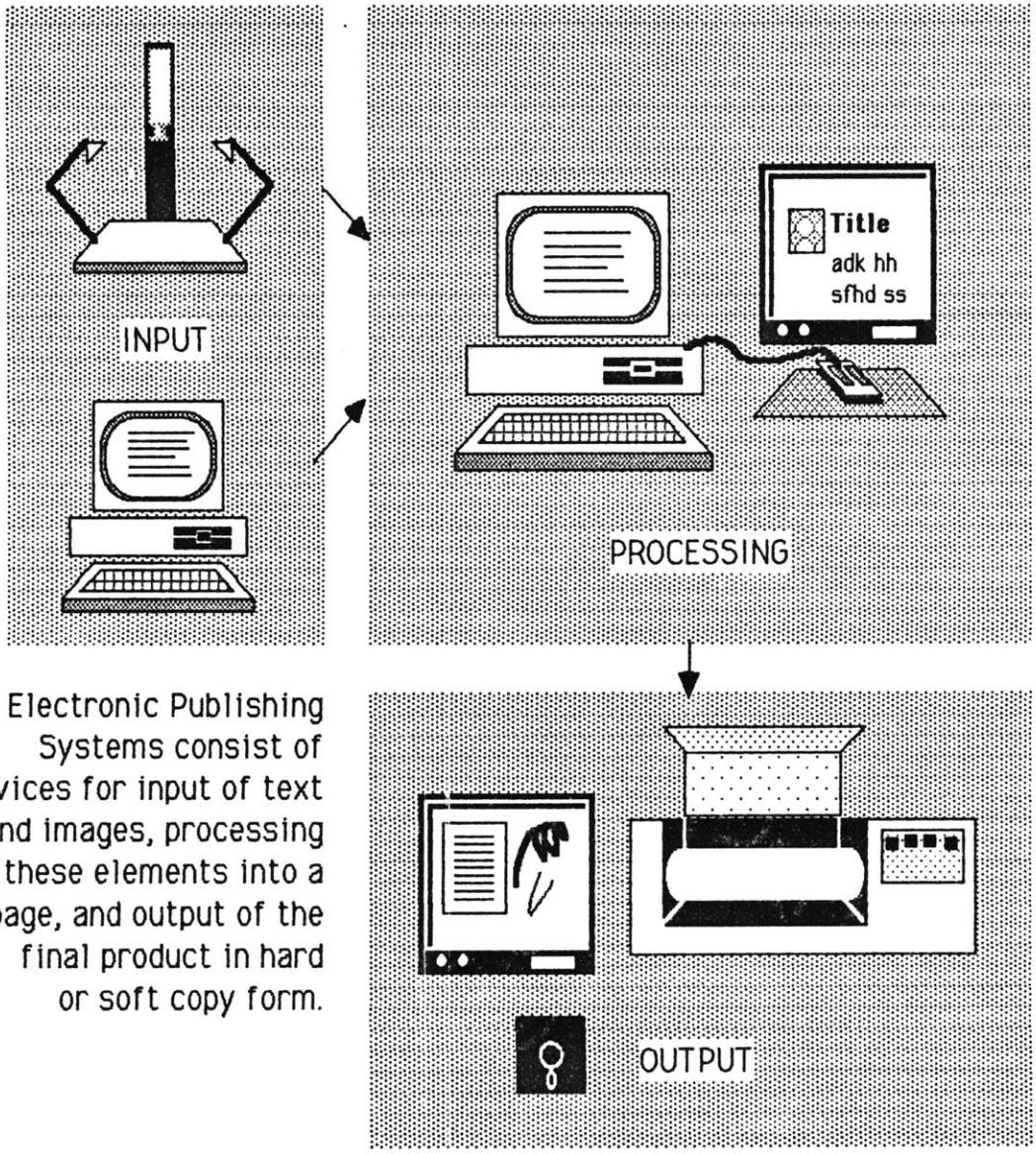


The very purpose of our alphabet is to give a visual structure to experiences, memory and abstract thought.
Will Burtin

The visual structure of information is just as important as its linguistic one in transmitting messages. One can support or influence the other. A notable example of this integral relationship between visual structure and text structure is Karl Gerstner's book "Compendium for Literates." Being both author and designer, he was able to produce a work where the visual unit is also a unit of meaning.

The design of information could be improved if the means of relating the linguistic structure of the information to a visual form was available to both authors and designers.

Electronic Publishing Systems



Electronic Publishing Systems consist of devices for input of text and images, processing of these elements into a page, and output of the final product in hard or soft copy form.

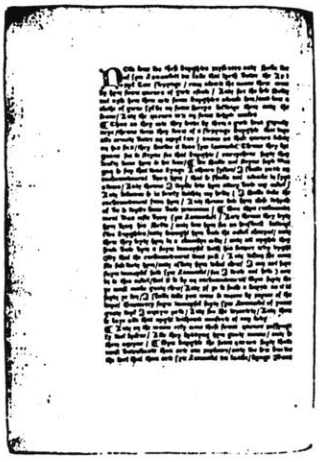
Electronic Publishing Systems

the combination of computer, telecommunications, and display technologies

Peter Zorkoczy

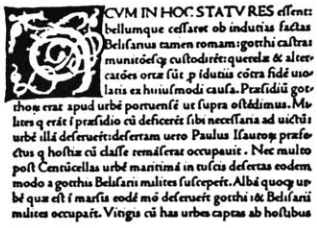
Four major revolutions have occurred that affect how we communicate. Writing allowed us to transmit information over space and time, printing provided the means to reach a mass audience through multiple reproductions of information; telecommunications and broadcast gave us the ability to communicate almost instantaneously over long distances, and the computer made it possible to manipulate, store, and retrieve information quickly and in greater volume. Today, all modes of communication are being brought into one grand system by electronic technology. Both the manipulation of symbols in computers, and the transmission of those symbols electrically or optically are being used at crucial stages in the production and distribution of information.

Electronic publishing has replaced many of the traditional means of production and distribution of information. Printing is moving from a mechanical process to an electronic one through the use of lasers and computers. On-line database systems such as the SOURCE and the Dow Jones News/Retrieval service distribute information on request for display on a CRT screen. Even the process of creating words and images is accomplished through word processors or computer-aided design systems. Greater efficiency and productivity, savings in time and materials, and the ability to make corrections painlessly are the impetus to this change.



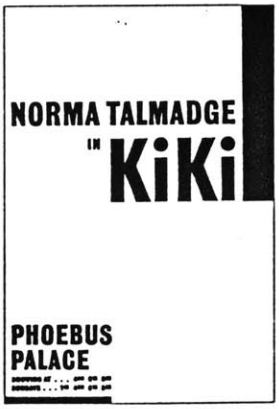
Page from
Sir Thomas Malory's
Le Morte d'Arthur
printed in 1485
by William Caxton.

From 1456, when Gutenberg printed his 42-line Bible using movable type, until the development of the steam-powered press in 1810, few advances were made in printing technology. The standard of the classic page, composed symmetrically and horizontally with an even grey tone, exemplified by *De Evangelica* printed in roman type by Nicolas Jenson in 1470, also underwent few changes until the early 1900's.

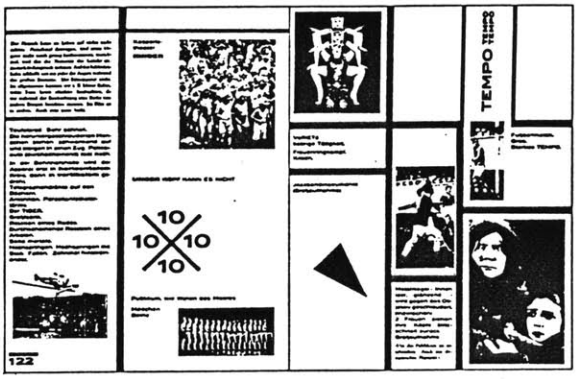


A portion of a page
from De Evangelica
printed in 1470
by Nicolas Jenson.

The turn of the century brought with it the machine age and a demand for modernism. Rejecting the purity of the "good grey page," the artists and writers of the day sought to imbue it with the principles of modern production: segmentation and the assembly of elementary forms. Jan Tschichold contributed to this with his theories of dynamic typography and asymmetric structure [TSCHICHOLD], as did Piet Mondrian through the pure division of space [SPENCER]. Greater mechanization in both printing presses and typesetters, plus the introduction of photographic processes also contributed to the liberation of the page from symmetry.



Poster design by Jan Tschichold, 1927, shows asymmetic typographic arrangement.



A double-page spread from Malerei, Photographie, Film, by L. Moholy-Nagy, 1925. The geometric division of the page is similiar to paintings by Piet Mondrian.

As printing technology became more complicated, and the demand increased, the creation and production tasks became more compartmentalized. Today, the publishing field is highly specialized, requiring numerous skilled technicians, artists and writers to create and produce a publication. Technology of the industrial era seems to have bred specialization and segmentation, as evidenced by Ford's auto asembly line. The result has been the lack of ownership and commitment by those involved in production, and the inability to look at a problem globally. However, today's technology may be able to bring us full circle to the point where our tools are so highly developed that they no longer require specialists to operate them. With the proper tools and environment, any one can be their own publisher, even the intended receiver of information.

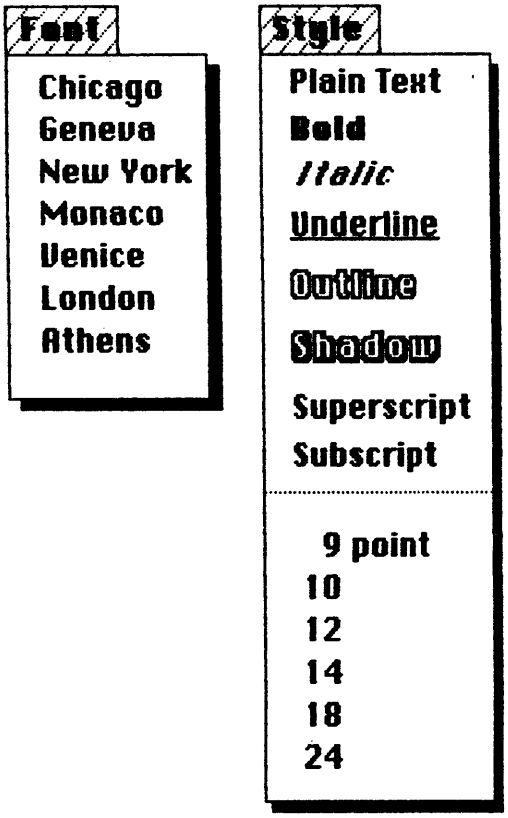
There is a wide range of electronic publishing tools available today, from text editors for personal publishing to sophisticated composition and scanning equipment for magazine design and production. Over the last 5 years, improvements have been made in the simulation of the end product (WYSIWIG displays), as well as the user interface. These electronic tools are extensions of more familiar mechanical or film-based ones for visualization and planning. They provide quick and easy visualization of ideas through direct manipulation of the page elements by the user.

Xerox Star 8010 Information System

The Xerox Star system, a professional office workstation released in 1981, established a standard for user-interface design that can be seen in most personal and office publishing systems available today. Through an iconic interface that employs a physical office metaphor and windows, objects and actions for document creation are made visible to the user. Functions or behaviors for integrating text and graphics can be accessed at any time by using a two button mouse to point at and select or adjust the appropriate object: text editor, graphics editor, or desktop. No memorization of format codes is required as property lists for objects can be accessed and changed through pop-up windows. The ability to select a text string, choose its appearance (bold, italic, underlined, superscript, subscript, larger font size, or smaller font size) from a property sheet, and see the result displayed on the screen is a vast improvement from text editors requiring the user to learn and remember keyboard coding instructions.

Following are example typeface menus from Apple's MacWrite application for the Macintosh, and an example of the control sequences used in TEX, a text formatting language developed by Donald Knuth.

Pull-down menus for selecting text properties from MacWrite



TEX coding examples for changing fonts in an existing file to produce the typeset sample below

```
to be \bf bold \rm
or to \sl emphasize \rm
something

or

to be {\bf bold}
or to {\sl emphasize}
something
```

to be **bold**
or to *emphasize*
something

The object-oriented approach and "what you see is what you get" (WYSIWIG) display principles of the Star system have been incorporated and extended in subsequent systems such as Apple's Macintosh computer and Interleaf's Office Publishing System.

In general, a complete electronic publishing system consists of hardware or software tools to support planning the job; entering images and text into the system; processing, creating, and assembling elements of the document, and producing the end product through various output devices. Systems are designed for specific types of publishing jobs that range from black and white personal newsletters to full color commercial magazines. Their complexity is determined by the amount of control and level of resolution the publishing task requires.

Personal publishing systems like the Macintosh consist of a WYSIWYG workstation with the ability to handle both text and graphics. Separate applications are used for creating text and graphics, and then merging them together on a page. Specification of typographic parameters consists of choosing a font and its style. Output is to a low resolution dot-matrix printer. Laser printers are also available, but currently at a cost too high for personal publishing.

Office publishing systems are only slightly more sophisticated in their ability to integrate text and graphics, and produce higher resolution output on laser printers and typesetters. The primary difference is in additional composition features such as kerning and inter letter/inter word spacing that provide greater typographic control. Image input capabilities are also added with electronic video cameras or scanners (CCD, laser, or photo diode).

Design and pre-press systems for the printing industry require a high degree of control and specification for producing color film separations or engraving gravure cylinders for printing. A variety of user-interface devices (mouse, tablet, keyboard, function keyboard, etc.) are incorporated into the system to give the user control of the fine details. Input and output systems are of the highest resolution.

Both office systems and production systems use different workstations for different tasks: writing, illustrating, and page makeup. Most use a monospace display or front-end composition system for text input and editing. WYSIWYG displays are used in workstations for creating illustrations and page assembly.

SCITEX makes two complete systems, one for design and the other for production. Vista interfaces to a front-end composition system and input/output scanner. It is the only designer's workstation to support a color WYSIWYG display and a full library of soft fonts. Publication designs created on the Vista can be digitally sent to the Response 300 for preparation for printing, or proofed on an Autokon scanner/proofer.

No system, no matter how simple or complex, accomplishes complete integration of text and graphics with one application. A "cut and paste" operation is still required to assemble text from a word processor, illustrations and and business charts from a graphics package, and photographic input from a scanner.

Being able to see the separate elements that make up a page is important in making decisions about selection, organization, and juxtaposition. Though most systems provide verbal or iconic lists from which to choose images or text, none except the Macintosh with its SCRAPBOOK

facility provide the ability to see what you have to work with at a glance. Even the SCRAPBOOK application is limited; only a portion of one image or text block is viewable at a time. For instance, placing this paragraph in the SCRAPBOOK, I could read up to the phrase "... only a portion of one image or text block is viewable at a time." Other images and text blocks can be seen by scrolling, but scrolling within a image or text block is not supported, and neither is the ability to increase the window or scale the image within it.

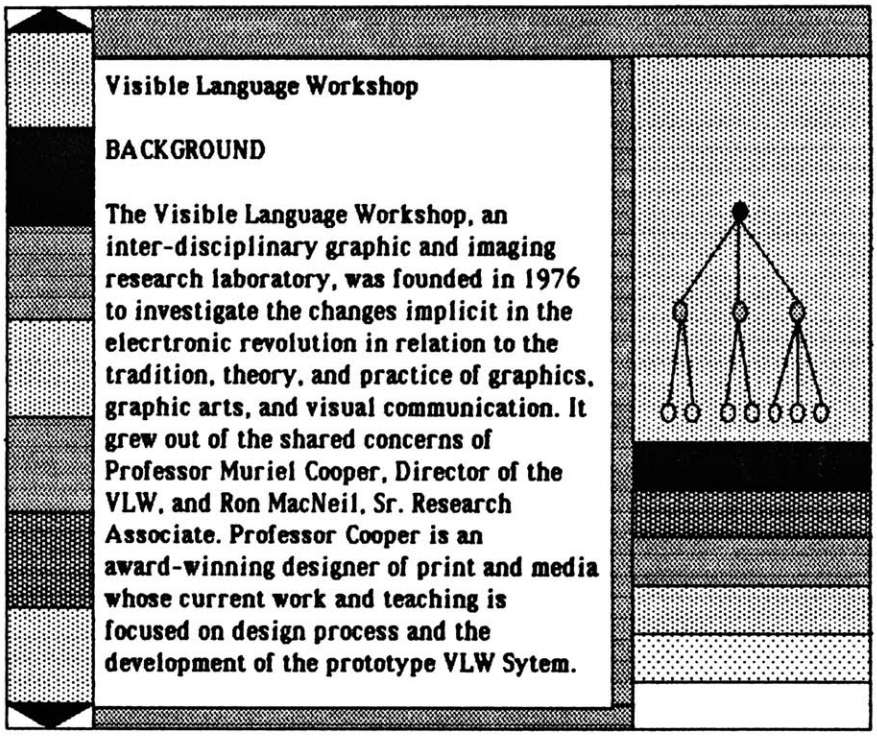
Though text is structured hierarchically, systems that process and format it treat the text as a linear structure to optimize processing speeds. In so doing, electronic publishing systems don't exploit the computer's ability to associate symbols and manipulate them based on their relationships. Most operate as batch pagination systems with Interleaf and Macintosh being examples of systems that process changes on the fly (close to real time).

The computer requests direct intervention by the user in making most of the higher level decisions about formatting (number of columns, pagination conflicts, type specifications, etc.), either through query (Texet) or property sheets. Some systems provide default property values where the user has not explicitly specified a value. None have rules or the ability to create them to propogate values from existing ones.

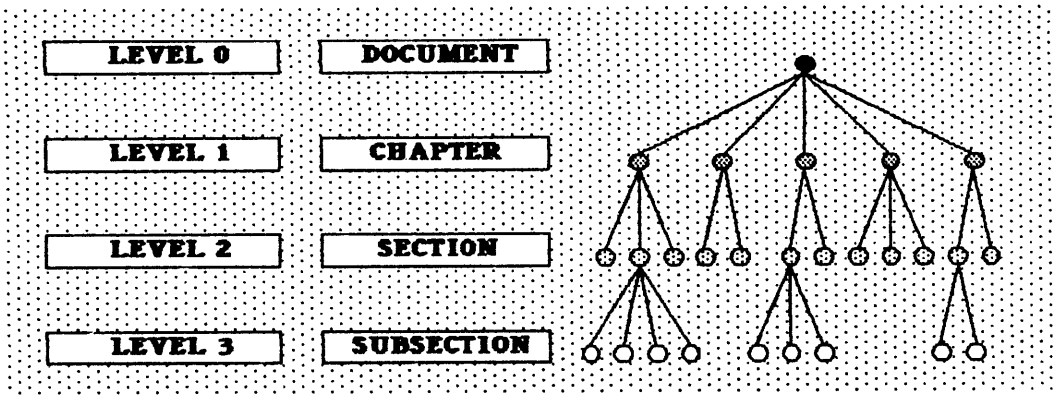
One approach to developing a rule-based system for document formatting is to use a hierarchical frame representation for the objects that the rules will act on. CONFORM is a step in this direction. It uses a recursive tree structure to store information about the text that is used in generating a typographic style sheet.

The next step from electronic tools is intelligent assistance in choosing the parameters for visualization. CONFORM establishes a framework to make this possible.

CONFORM

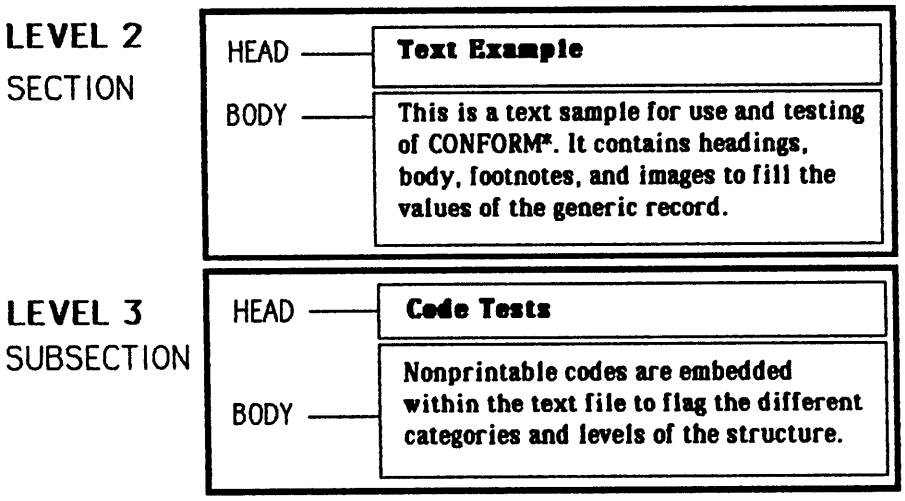

Visible Language Workshop
BACKGROUND
The Visible Language Workshop, an inter-disciplinary graphic and imaging research laboratory, was founded in 1976 to investigate the changes implicit in the elecrtronic revolution in relation to the tradition, theory, and practice of graphics, graphic arts, and visual communication. It grew out of the shared concerns of Professor Muriel Cooper, Director of the VLW, and Ron MacNeil, Sr. Research Associate. Professor Cooper is an award-winning designer of print and media whose current work and teaching is focused on design process and the development of the prototype VLW Sytem.

Functional Desciption CONFORM is a method for chunking and organizing the information contained in publications that integrate text and graphics. Text categories and images are selected by the user and automatically related to one another in a tree structure.

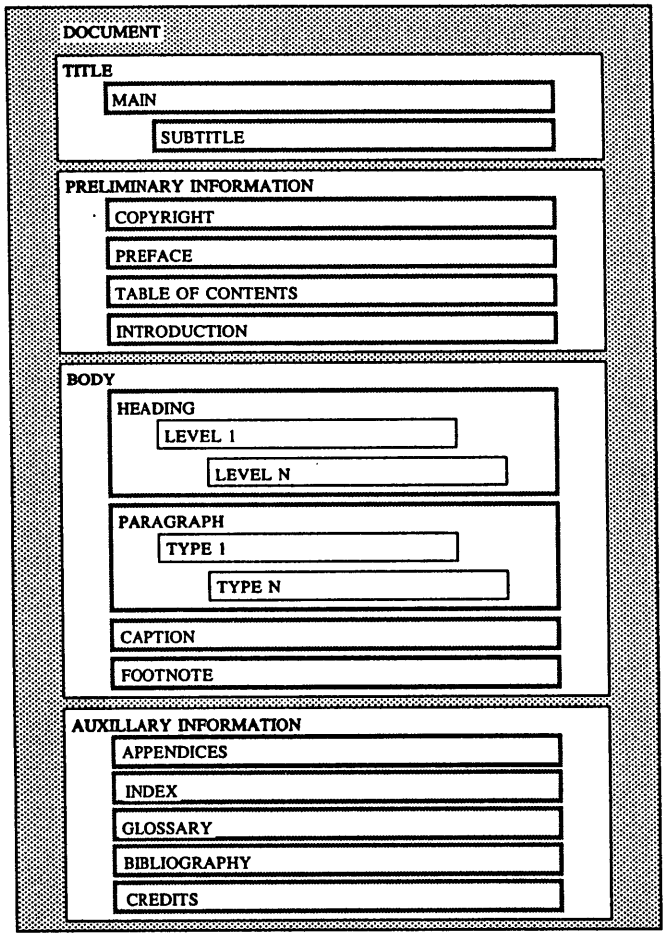


Placing the elements of the document in such a context sets the framework for generating solutions based on their relationships. Information about the tree can be used in generating the typographic style sheet, accessing a particular part of the text, or generating an outline. CONFORM establishes a basis for developing a rule-based system that will assist a designer with his or her task. The aim is to use the content analysis of the document as a means for determining its final form.

Content is defined through quantitative analysis of a manuscript by the user. This analysis involves reading the text and identifying the levels of information contained in it (document title, chapter title, section heading, subsection heading, and so on). Defining content in terms of levels of information or text categories, and relating them in a tree data structure provides a powerful abstraction that the computer can deal with in assisting in the design of the document.



In defining text categories, I am referring to the hierarchical structure of the text; its title, subtitle, chapter headings, subheads, body, captions, footnotes, etc.



This diagram illustrates one possible example of text categorization. Most, if not all, documents consist of the top level categories (title, preliminary information, body, and auxillary information), but their succeeding levels may vary. In actual use, the categories under preliminary and auxillary information would consist of additional levels similar to those under body.

Parsing the manuscript into levels of information is a lot like diagramming a sentence, it creates a map to guide the reader through the information. We naturally chunk information in order to remember and make judgments about it. Relating the verbal structure of our messages to a visual structure supports meaning and helps in understanding it.

Categories may be assigned by a designer who is planning the layout of a document, or by the writer of the document while creating and editing the text. If the manuscript has been chunked by the author, this information may be used as an outline by the designer along with their own analysis and interpretation of the text for further segmentation of the document. Ultimately, we would like the computer to provide us with a first pass chunking of the text, but that requires understanding what was read.

Creating dependencies and relationships through a hierarchical text structure gives us the means to apply rules or instructions for displaying the text in appropriate spatial and typographic parameters; appropriate to the message and to the reader of the message. For designers of information, separating the structure from the visual representations provides greater latitude in making and visualizing changes. The text structure can remain the same while the actual visual representations related to it change either globally or locally. Changing the look of the document is easily accomplished by identifying in which category the typographic parameters are to change, or which parameter is to change independent of category.

A

Heading

Subheading
Paragraph of text. Parameters assigned to this chunk depends on its type. If it is a quote then it may be set in italic rather than roman.

B

Heading

Subheading
Paragraph of text. Parameters assigned to this chunk depends on its type. If it is a quote then it may be set in italic rather than roman.

C

Heading

Subheading
Paragraph of text. Parameters assigned to this chunk depends on its type. If it is a quote then it may be set in italic rather than roman.

In example A, heading and subhead are both in bold. Example B illustrates changing the bold typographic parameter to bold italic which affects both heading and subheading. Changes can also apply to a specific text category instead of the entire document as illustrated in example C.

In a WYSIWYG system like the Macintosh, changes in style require reselection of the text string needing to be changed. This "painting" of typographic parameters can be tedious for long documents. Other systems use markup codes that are added to the text via the keyboard to distinguish its structural elements, e.g., running heads, footnotes, quotes, tables, etc. The Association of American Publishers is currently working on a project that will establish a generic coding standard so that electronic files are transferrable between machines, and can be processed or used as input for any application. The issue in this thesis is not so much what the codes are, but how the user identifies a text's structural parts, how they interact with the computer, and how the computer understands or knows what the structural parts are.

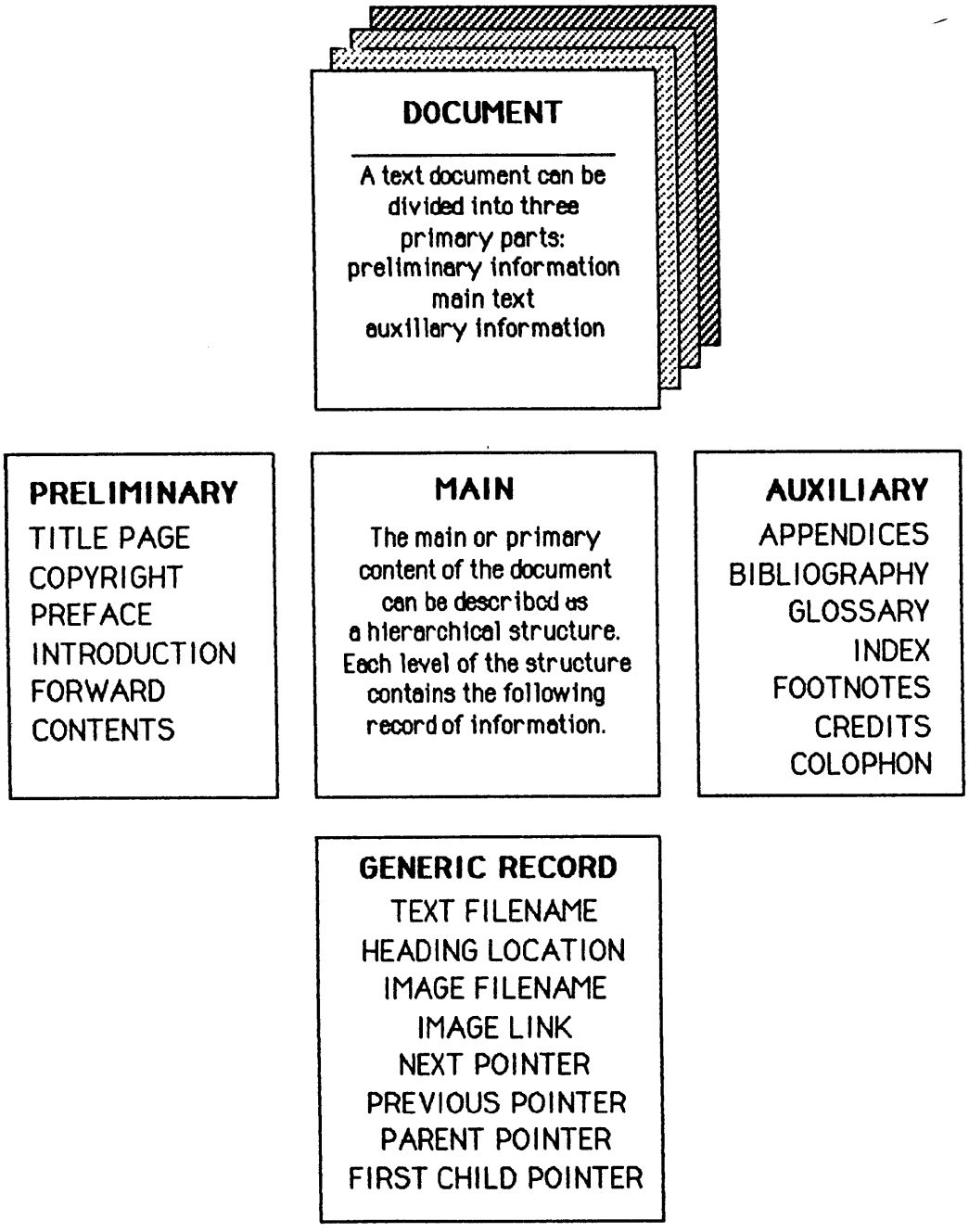
One of the objectives of CONFORM is to provide a user interface that makes the process of chunking the text and relating images to it as natural as marking a paper manuscript with red pencil. Of primary importance, the user should not have to learn a new verbal language or memorize codes to indicate the object or action he or she wants performed.

CONFORM creates an environment for the user to communicate with graphically and gesturally. A mouse is used to point at and select objects or actions. Feedback to the user is given graphically by adding marks in the form of color boxes, changing the color of an object, or opening new views. Using the mouse to select text strings is similar to using a pencil to underline words in a manuscript, except in this case the computer makes the mark.

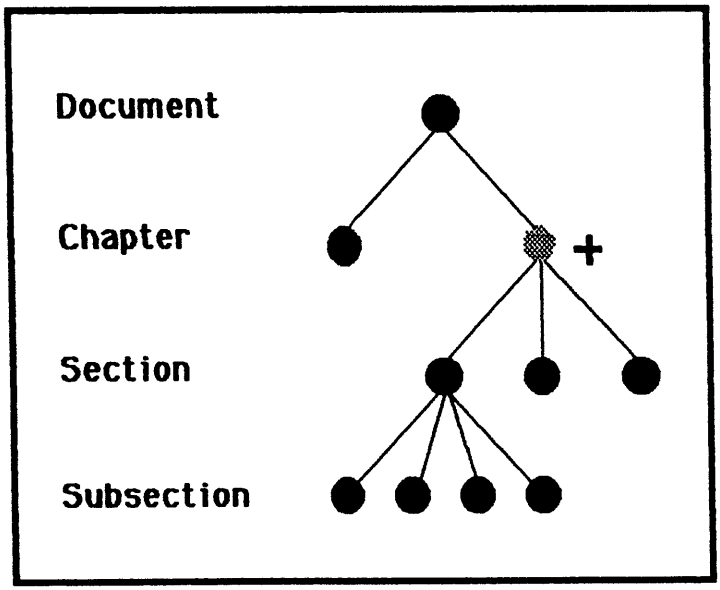
To indicate that a text string has been selected, it is rewritten in a different color that corresponds to a text or image category. Color in varying intensities is used to indicate the levels of the text instead of some other means, like changing weight or size, so that there is no confusion on the part of the user that the operations being performed are to indicate levels of information, not how they should appear.

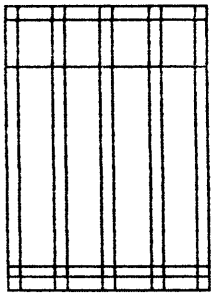
As the user identifies the structure of the text, information about the categories, such as the filename the text is found in, the selected string's offset into the file, and any images associated with it, is stored in a tree data structure. Nonprintable codes are also embedded in the text file to flag the category type of the string that follows. If the file is ever edited, the tree can be recompiled from these codes.

Following is a representation of the linguistic hierarchical structure of a document and the generic record that is used to describe it.

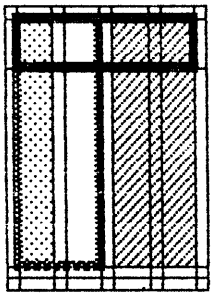


A graphic representation of the tree is displayed to provide the user with an abstract view of the document. As text strings are selected and categories defined, the tree adds a new node and redraws itself. The tree structure and its representation can also be used to access different parts of the document. Selecting a tree node with the mouse, the user can get the information contained in that node (the actual text string it represents), redisplay the text on the screen starting at that location, or select the part of the document he/she wants to preview in page form.

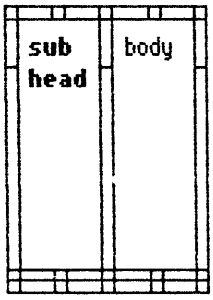


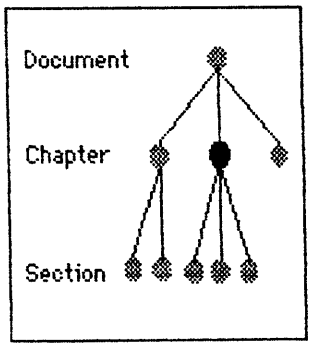


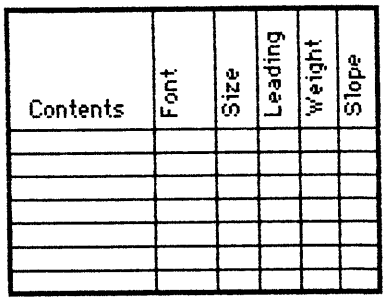
The form of a document is described in terms of page constraints, image styles, and typographic parameters.

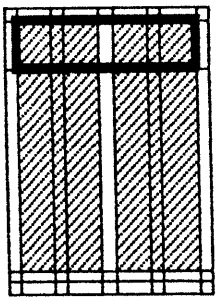


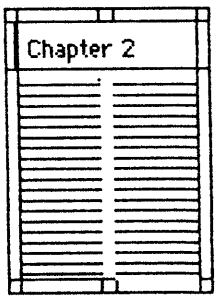
CONFORM uses a spatial representation of a page that is described through grids and regions. A grid is a mechanism used in publication design to spatially divide a page into columns and margins, and to provide for alignment of its elements (text and image). Usually one grid is developed for an entire publication with variation created through the use of regions built on top of the grid. Regions may span more than one column plus overlap each other. Depending on the specific text and image elements to be arranged, certain regions are activated while others are deactivated. This is determined through the use of region property lists that contain information about the type of element that goes in it.











After a manuscript's content has been defined in CONFORM, the content analysis represented by the hierarchical structure is used in a typographic style sheet along with default rules, or rules created by the user, to generate the typographic parameters for the document. Style sheets that have been previously created and stored may also be used to format a document. To preview a page of the document, the user selects the part of the text he or she wants to view, and the text automatically flows into the appropriate regions in the typographic parameters defined in the style sheet.

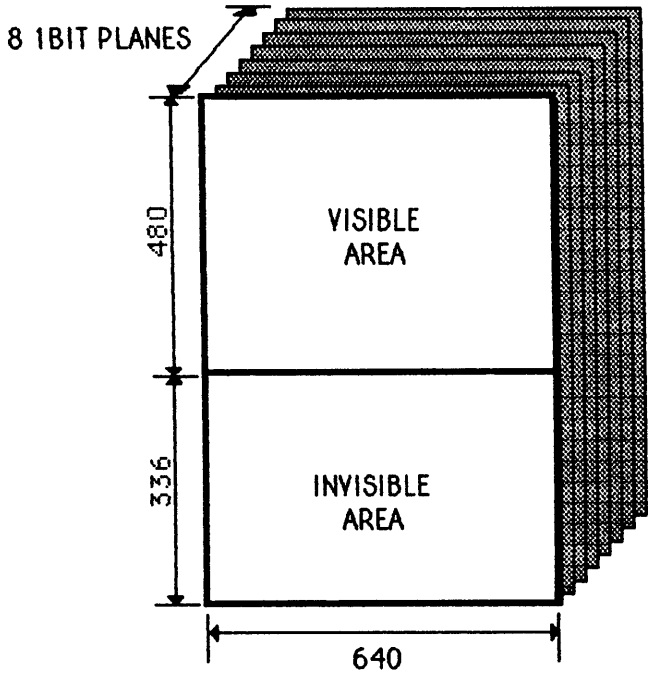
The next section provides a more detailed description of the data structure and functions used in CONFORM, and an example session is included to better understand the interaction.

Design and Implementation

The development environment for CONFORM consists of an IBM XT equipped with 512k RAM, a 10Mb hard disk, and an experimental graphics display adapter (YODA). An analog RGB monitor is used for color display of the task environment, and a three button optical mouse for user interaction. Implementation was done in the C programming language with the use of basic graphic routines supplied as part of the YODA graphics package.

A description of the YODA framebuffer and basic graphic routines supplies the background information behind CONFORM's design and development. An overview of the task environment provides the framework to discuss the program control and interaction. Recursive tree functions and data structures will briefly be described, as well as an example session to illustrate using CONFORM. More detailed software documentation can be found in the appendix.

YODA Framebuffer The graphics display adapter called YODA consists of a 640 by 815 by 8bits/pixel frame buffer with a viewable resolution of 640 by 480. The remaining 640 by 335 part of the frame buffer is used as an invisible area to store anti-aliased font information or graphics for copying into the visible area or. The user may have as many as 256 different colors/intensities in each frame chosen from a pallette of over 16 million possible colors. Video output is at 30Hz interlaced to an RGB monitor.



YODA Functions

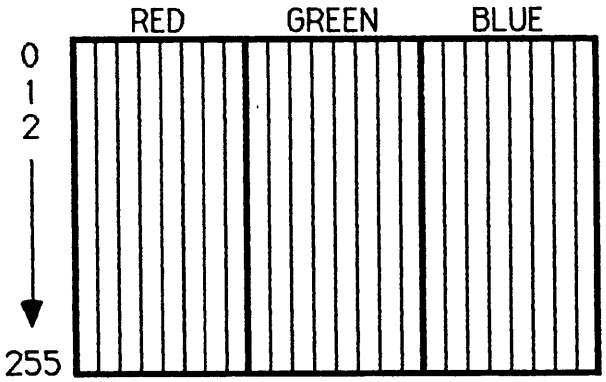
Following is a description of YODA concepts and functions important to understanding CONFORM's implementation.

PMD

The pixel map concept provides a means to create subareas within the frame buffer that can be addressed and written to in coordinates relative to the pixel map's origin. A **pmd** is a pixel map descriptor for a rectangular area in the frame buffer. This area is defined by an origin (x, y, z) and extent dimensions (width, height, depth), with the z-extent referring to the number of planes (0-7) in the pixel map - not a 3D space. The visible and invisible portions of the framebuffer are accessed through **pmd's** that are returned on opening the display. Other pixel maps are created in relation to either of these, or to any pixel map that has already been defined. All functions use a **pmd** to describe the area to write to, and automatically clip to its boundaries. CONFORM's screen environment is defined through the use of pixel maps.

Video Lookup Table

This is a two dimensional array of 256 colors with 8bits each RGB. Pixel values are not themselves the colorvalue, but an index into the array. Antialiased graphics and text require 16 grey levels interpolated between the background and foreground colors. CONFORM uses slots 0-196 for 12 color combinations for antialiased text.



Antialiased Text

The YODA graphics package provides serif and sanserif fonts ranging from 10pt to 48pt. These 4 bit fonts require 16 grey levels to suppress the stairstep effect of the raster display. A basic function is provided to interpolate between the background color of the text and its foreground.

To display the fonts, a portion of the invisible framebuffer must be set aside and initialized as a common font storage pool to manage the compressed character data. Basic functions are provided to open and load the fonts, set the position for writing, and then write a text string. The text position is defined as an (x,y) position relative to the pmd, and specified as two double integers to allow for subpixel positioning. The x value is rounded to the nearest 1/8th of a pixel, and corresponds to the leading character space. Subpixel positioning is not supported in the y direction. The y value is rounded to the nearest integer, and refers to the baseline position of the text. CONFORM builds on these basic functions, plus functions provided to access information about the font or the length of the text string, to print a page of text that has been read from a text file.

Setting up the Task Environment

Screen Environment

The screen environment created for the user consists of an image reference area, page area, and tree/menu area. These are defined as PMD's with subareas created in relation to them.

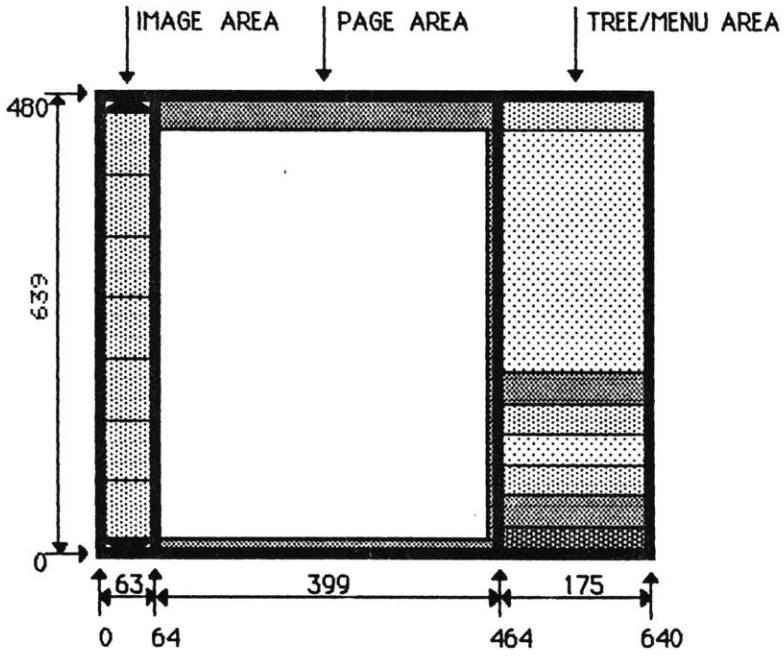


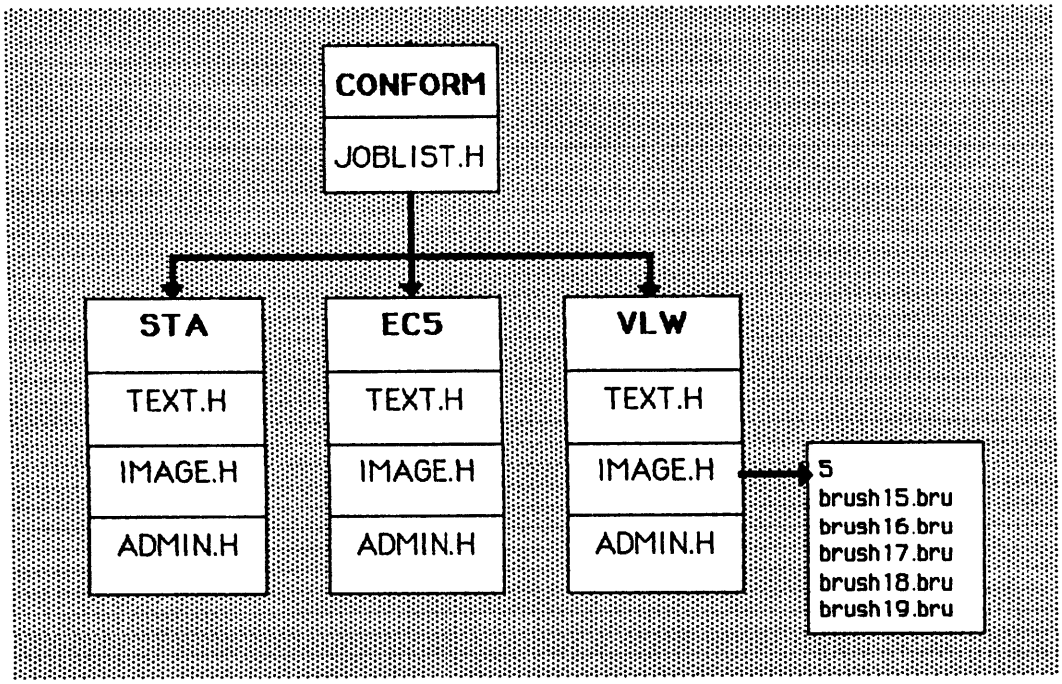
Image Reference Area — This space contains seven 64x64 slots for displaying scaled versions of images to be included in a manuscript. The top most slot is reserved for the generic image icon which acts as a place holder in the text for artwork that needs to be developed. Additional images can be displayed by scrolling the image area up or down.

Page Area — The largest screen area is reserved for displaying the text file using anitaliased fonts. To the right and bottom of the page are scroll bars for scrolling the text a page or line at a time. At the top is an area for the document name the user is currently working on.

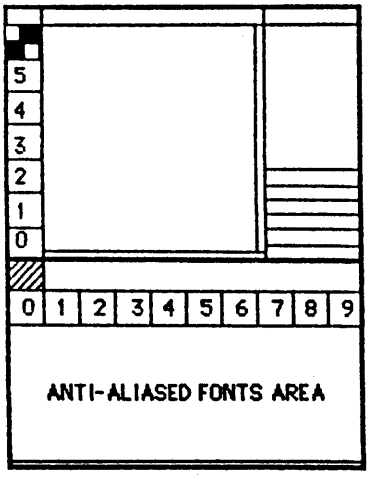
Tree/Menu Area — A list of menu options is provided for the user in order to access additional information or make changes. A graphic representation of the document structure is displayed and updated as the user interacts with the system. A description of the menu options can be found in the section describing program control and interaction.

File Management

To begin, each document to be designed is given a job name under which all the information concerning it is organized. A job is defined as a directory containing text, image, and administrative files about the document. Each file is listed in one of three header files (text.h, image.h, or admin.h) which contains the number of files of that type and the filenames under which the information is stored.



CONFORM's initial task is to retreive the list of jobs currently in the system from "joblist.h", and display the names so the user can chose which one they want to work on. Once a job is selected, the system retreives and loads the appropriate files into the task environment.



The list of images associated with the job is retreived from "image.h". Each image file is opened and loaded one at a time into the visible portion of the framebuffer, then scaled to fit within the 64x64 slot. Slots are numbered 0 through 5 with slot 0 beginning at the lower end of the reference area. If there are more than 6 images, the remaining are copied into the invisible portion of the framebuffer. The adjacent diagram shows the numbering and distribution of the image slots.

A document's manuscript may exist in one or multiple files. The first text file listed in "text.h" is opened and and the entire file loaded into the buffer. Additional files are opened and loaded as needed.

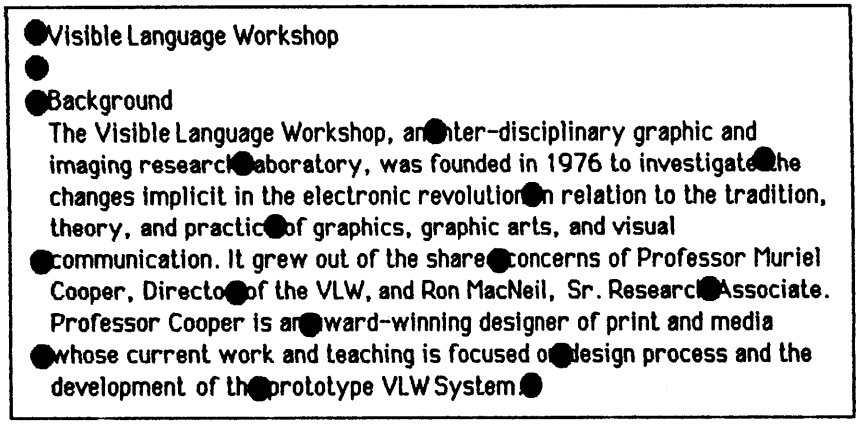
Two data structures are used to store information about the text and the region it will be written to, and to aid in mapping between the physical display and the text buffer.

```
struct text_region
{     pmd *area;
      int wd;
      int ht;
      int lmargin;
      int rmargin;
      int tmargin;
      int bmargin;
      int color;
}
```

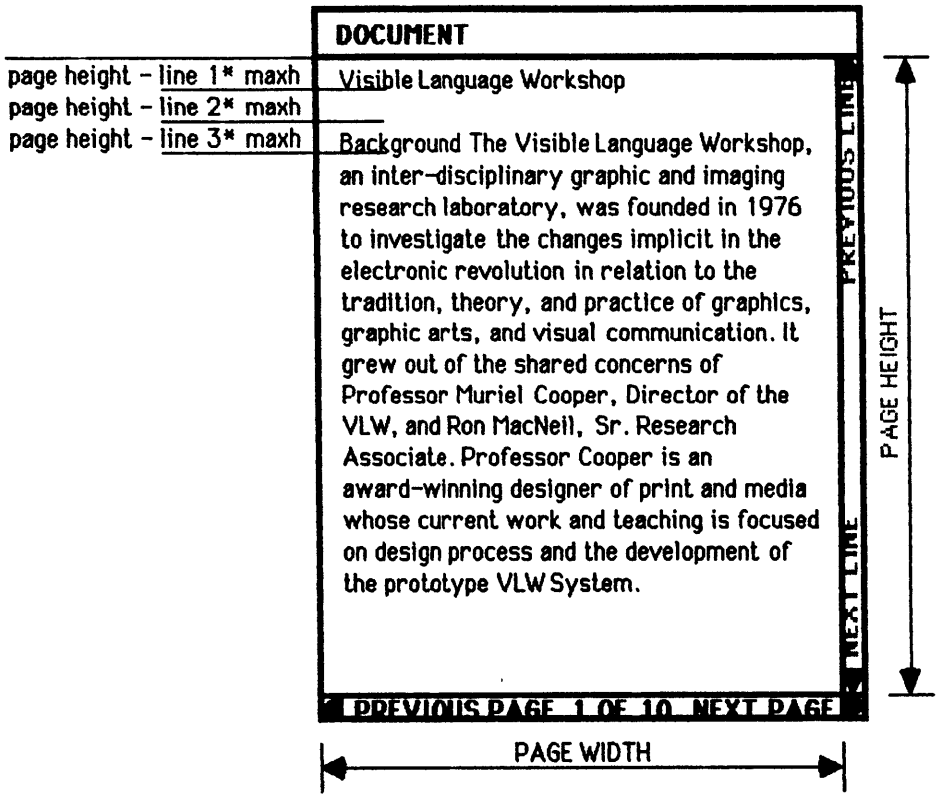
```
struct text
{     char fname[16];
      long linebr[512];
      int maxlns;
      int bsize;
      char *buf;
}
```

Line breaks are calculated for displaying the text in the page region using the antialiased fonts, and stored as offsets in **linebr[]**. This array is used as an index into the text buffer for printing the text to the display. The print page function collects a line to print by reading characters from the buffer starting at linebr[0] to linebr[1]. The first line is positioned at the left margin with the baseline positioned the height of the font down from the top of the page. Subsequent lines are collected and printed by incrementing the buffer index and the distance from the top of the page.

BUFFER

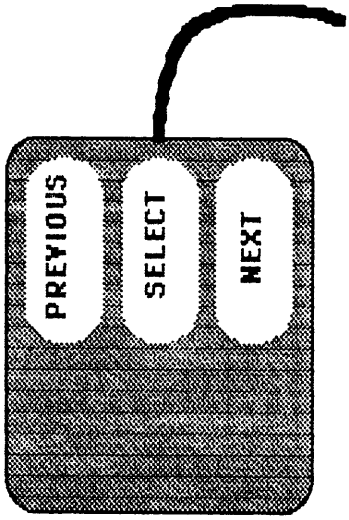


● Indicates offset location of the line breaks calculated for the display


DOCUMENT
page height - line 1* maxh
page height - line 2* maxh
page height - line 3* maxh
Visible Language Workshop
Background The Visible Language Workshop, an inter-disciplinary graphic and imaging research laboratory, was founded in 1976 to investigate the changes implicit in the electronic revolution in relation to the tradition, theory, and practice of graphics, graphic arts, and visual communication. It grew out of the shared concerns of Professor Muriel Cooper, Director of the VLW, and Ron MacNeil, Sr. Research Associate. Professor Cooper is an award-winning designer of print and media whose current work and teaching is focused on design process and the development of the prototype VLW System.
PREVIOUS LINE
NEXT LINE
PAGE HEIGHT
PREVIOUS PAGE 1 OF 10 NEXT PAGE
PAGE WIDTH

User Interaction

The interaction is designed to allow the user to freely choose among the items on the screen. Program control is accomplished through cursor location or context. The cursor routine **tab()** updates and returns the cursor's location and mouse button state. First the cursor is initialized (**initcurs()**) to the visible area of the frame buffer. Using the mouse, the user moves the cursor and presses one of the three buttons. Mouse conventions adopted for this application are SELECT, PREVIOUS or UP, and NEXT or DOWN (middle, left, and right respectively). These basic meanings are supported throughout, with the specific action dependent on the context, or spatial location of the cursor. For instance, in the image area the left or right button has the effect of scrolling the image reference space up or down, while in the page area they have the effect of moving up or down the tree.

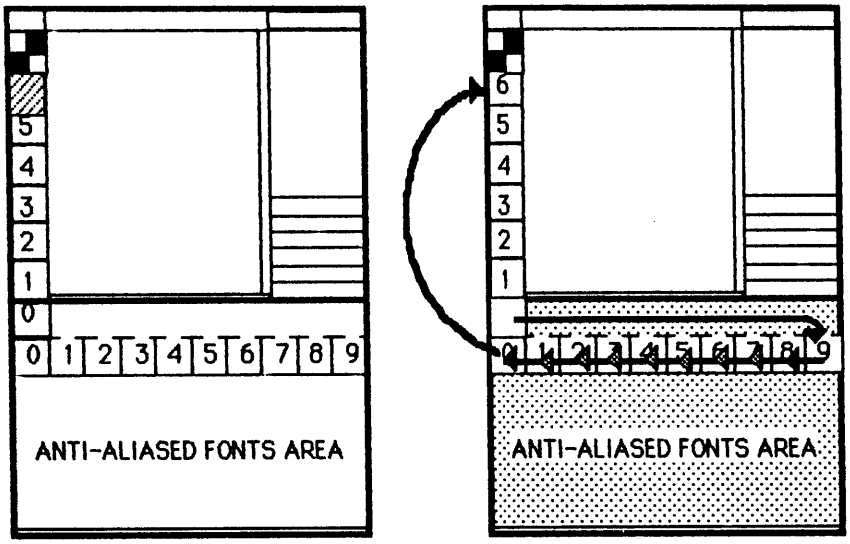


When any button is pressed, program control is invoked first by determining which of the three major areas the cursor is pointing to. This is accomplished by performing a check on the cursor's x value. Action is then passed to that area, and is determined by the mouse button state.

Image Area

PREVIOUS IMAGE AND NEXT IMAGE

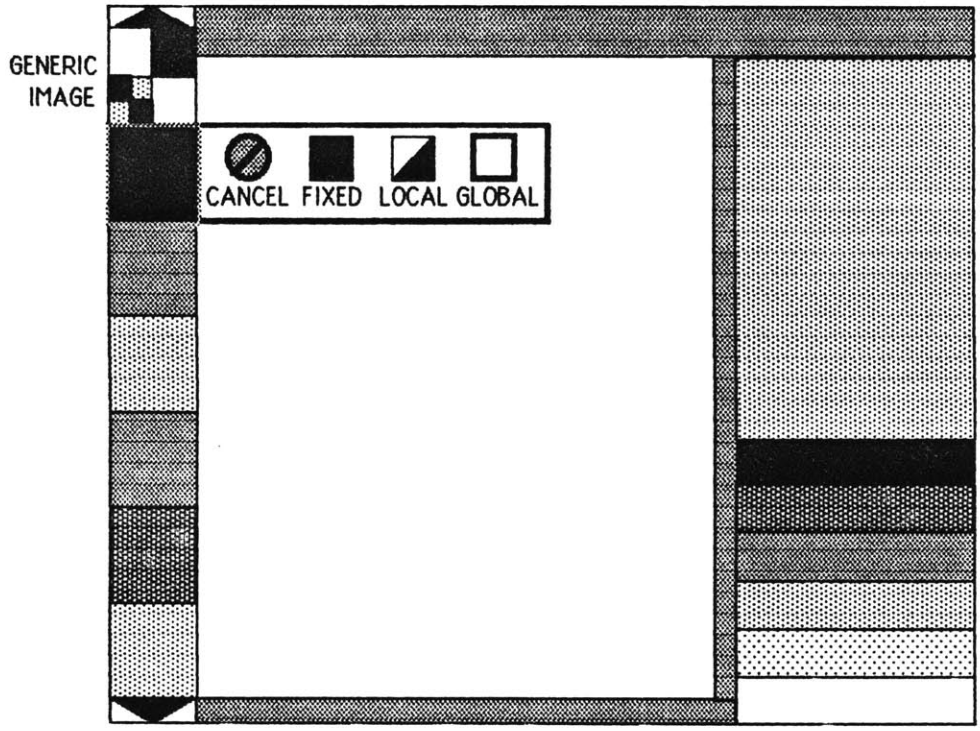
Image numbers are used as an index into the array of image names. These numbers are linked to the slot numbers of the physical display, but are separate from them. To keep track of which image is in which slot, the number of the image in slot #0 is stored in the variable **first_img**, and the number of the image in slot #5 is stored in **last_img**. When the left or right buttons on the mouse are pressed, the values of these variables are decremented or incremented, respectively, to correspond the image that is currently displayed in those slots.



To scroll the image bar the **BitBlt()** function is used to copy images from and to the invisible portion of the framebuffer. For instance, to see the next image in the list, the image in slot #0 is copied to the temporary slot in the invisible area. Images in slots #1-5 are then moved as a block to slots #0-4 by creating two pmd's and using the BitBlt COPY function. The next image in the list, which is located in invisible slot #0, is then copied to slot #5. To prepare the invisible image area for subsequent calls, images in slots #1-9 are shifted to slots #0-8, and the image in the temporary slot is copied to slot #9.

SELECT

Specific images or the generic image icon can be linked to specific text strings in the manuscript. The user selects an image by pointing to it with the mouse and pressing the SELECT button. The y value returned from **tab()** is used to determine which slot the cursor is pointing to, and the image number is determined by adding the slot number to **first_img**. To indicate which image the user selected, a red outline box is drawn around it.



The cursor is then placed in the page area for selecting the text string the image is to be associated with. Once the string is selected, a menu pops up along side the image for the user to select the flexibility of the link, or cancel the selection entirely.

Page Area **SELECT**

While the cursor is in the page area, the SELECT button can be used to scroll the text or select a text string.

Scrolling the text is similiar to scrolling the image reference area. The display line numbers along with the value for first line (**firstln**) and next line (**nextln**)are used to index the text buffer. To scroll the the text, **firstln** is incremented or decremented either by 1 or the maximum number of lines on the display (**maxdepth**). This new value is used as the index into the the line break array (**linbr[]**). The value of **linebr[firstln]** is the offset into the text buffer to begin reading characters for the first line of the display. Characters are read into

line[] until the current offset of the buffer equals the value of the next item in **linebr[]**. Each line is collected and then written to the display until the number of lines written equals **maxdepth** or the buffer is exhausted.

A double SELECT operation is used to select a word or a string within the visible page. Text strings are selected by pointing to the first or last word in the string and clicking (press and release) the SELECT button, and then pointing to the other terminus of the string and clicking the SELECT button again. A single word is selected by pointing to it and clicking the SELECT button twice.

PREVIOUS LEVEL or NEXT LEVEL

Level refers to the the depth of the tree, and corresponds to a level of information. Selecting text strings is a means of defining levels of information or linking images with the text. Levels are created and accessed through the mouse buttons: the left button to go to a previous level, and the right to move to the next.

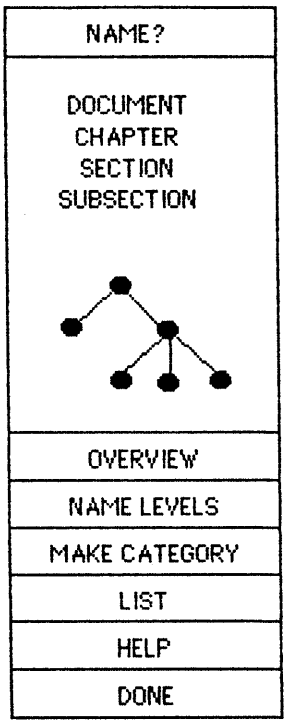
When the program is in its start state, the first text string chosen is the top level (document level) of the structure. As subsequent strings and levels are selected a graphic representation of the structure is drawn as an inverted tree with the last item created highlighted. Information about the tree is given in the Document Structure section.

Menu Area

The only button which is active in the menu area is the SELECT button. Following is a description of the menu options that may be selected.

OVERVIEW

Opens a window that displays the entire tree with its labels to provide an abstract overview of the document. Nodes can be selected and their associated text strings displayed. This is also used to move to another part of the text or select the chunk to be previewed in page makeup form.



NAME LEVELS

User defined labels for each of the tree levels are automatically defined when a new level is selected. A default label is flashed on the screen, which the user can change by typing in a new name, or clicking the SELECT button on the mouse to retain the default. NAME LEVELS is intended to allow the user to define all the labels before working on the document, or to change the labels at any point in the process.

MAKE CATEGORY

Allows the user to create their own category for identification in the text, such as glossary words, quotes, footnotes, etc.

LIST

This option allows the user to list the text strings that have been selected. Two lists are provided, a Document Outline or Image List.

HELP

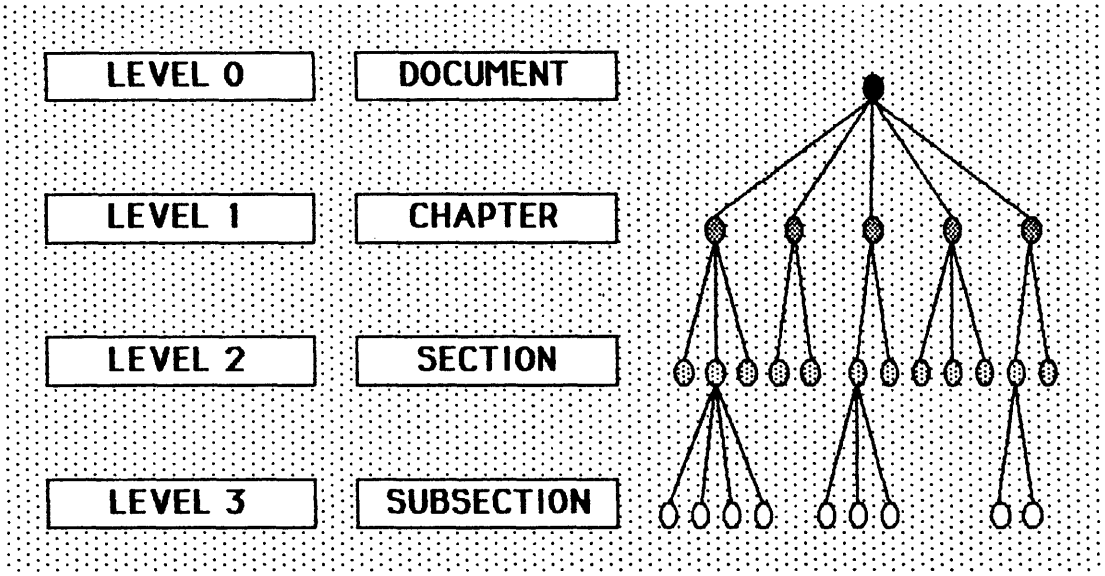
To provide assistance to the user, a short description is given about how to flag the text, link images to it, name the tree levels, and what the menu options are.

DONE

Selecting this item ends the session. Currently, the text represented by the last selected tree node is previewed in a format described by the typographic style sheet and spatial representation.

Document Structure This section will describe the tree structure used to organize the document, and the embedded codes used to flag the text. As background to this, the text categories used to define the document's structure will be explained.

TEXT TREE STRUCTURE



Text Categories There are two basic text categories defined in CONFORM plus a category for images. These are defined as symbolic constants **HEAD**, **BODY**, and **IMAGE** that have the ascii character values 128, 160, and 192, respectively. Levels within a category are indicated by incrementing these base category values.

HEAD category levels refer to levels in the tree and correspond to heading types, such as title, heading, subheading, or body heading. **IMAGE** levels refer to the flexibility of the link between an image and its associated text string, whether the image must be located with the string, or can be placed locally or globally within the document. Incrementing the base value for **BODY** is done to create user defined subcategories, such as glossary words, footnotes, or captions.

LEVEL 0	DOCUMENT	FLAG TYPE
	TITLE	**HEAD**
	SUBTITLE	**BODY**

LEVEL 1	CHAPTER	FLAG TYPE
	HEADING	**HEAD**
	INTRO	**BODY**

LEVEL 2	SECTION	FLAG TYPE
	SUBHEADING	**HEAD**
	BODY	**BODY**

LEVEL 3	SUBSECTION	FLAG TYPE
	BODY HEADING	**HEAD**
	BODY	**BODY**

Embedded Codes

When a text string is selected, information about it is stored in the tree node, and a nonprintable ascii code is embedded into the text buffer. These codes act as flags to the different categories and levels of the structure. ASCII characters 128 through 255 are used for the nonprintable codes. Each category is assigned a base character represented as an octal number.

CATEGORY CODES	HEAD '\200'
	BODY '\240'
	IMAGE '\300'

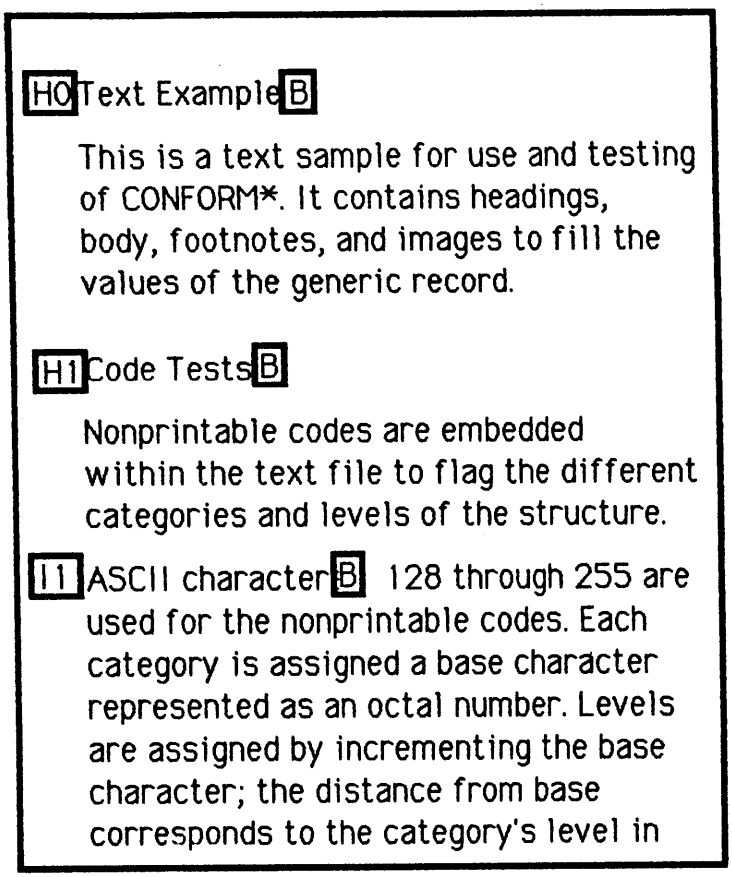
Levels are assigned by incrementing the base character. The distance from the base corresponds to the category's level in the structure. Each category can be assigned up to 32 levels.

HEAD is a base value that is incremented as the user moves down the tree to select subordinate text strings. When new categories are made by the user, they are assigned a name and ascii value incremented from **BODY**.

The flexibility of the image link associated with a text string is represented in the code in a similiar manner by incrementing from the base value.

IMAGE CATEGORY	'\300' RIGID
	'\301' LOCAL
	'\302' GLOBAL

These category codes are used in marking up the document, and displaying the marked text strings in a color related to their category and level. The appropriate heading level code is embedded at the beginning of the string, and a body code is embedded at the end. If the string being selected is for images, the image code is embedded at the beginning of the string and body at the end. When reading the structured document for formatting, these codes act as flags to retreive the appropriate instructions from the system.

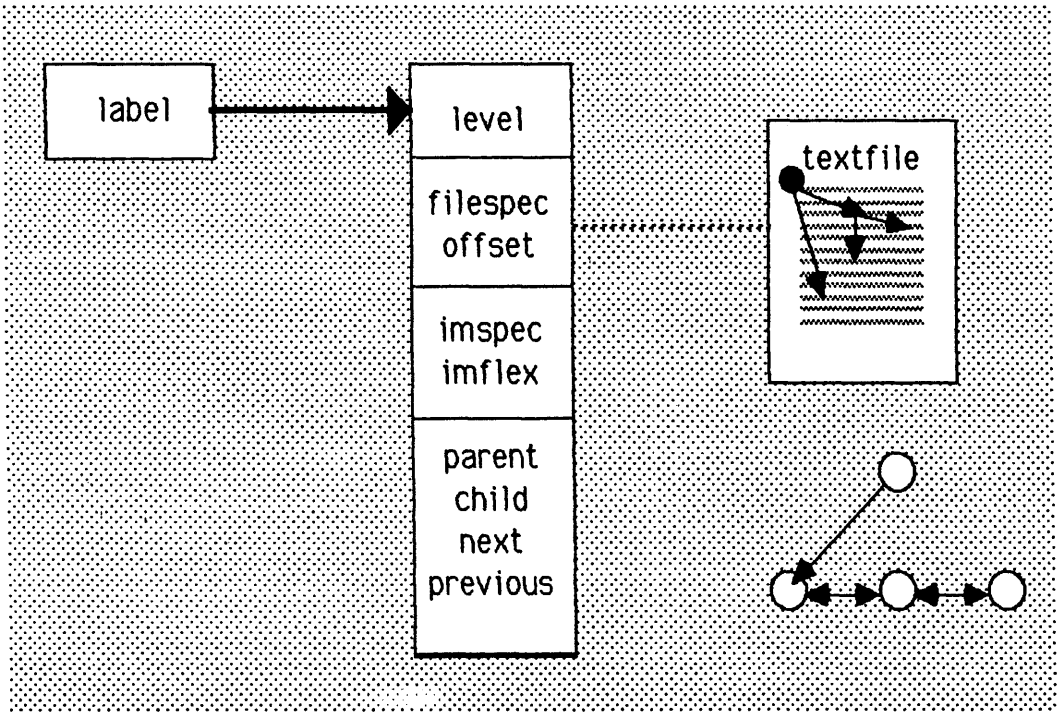


The above illustration depicts the location of the codes in the text. *Text Example* is the main heading and *Code Tests* is its subordinate. The text string *ASCII character* is associated with an image. The flexibility of the link between the image and text is 1, which means the image can appear locally within the text segment associated with *Code Tests*.

Embedding codes into the text file is not a new idea. Most systems have some means of entering format codes or image positions into the text. The Xerox Star system uses an "anchor" character for positioning graphics within text. As the text is edited the codes still have the same association. CONFORM's combination tree structure and embedded coding scheme provides greater latitude for making changes. If the text is edited after structuring it, the tree can be recompiled from these codes.

Tree Structure

A recursive tree data structure allows the document structure to grow to any breadth or depth as long as there is memory to support it. Each node in the tree represents a level of information in the text, and consists of these fundamental parts:



Operations that extract information from the tree are implemented as functions that return values. These functions are listed in the software documentation found in the appendix.

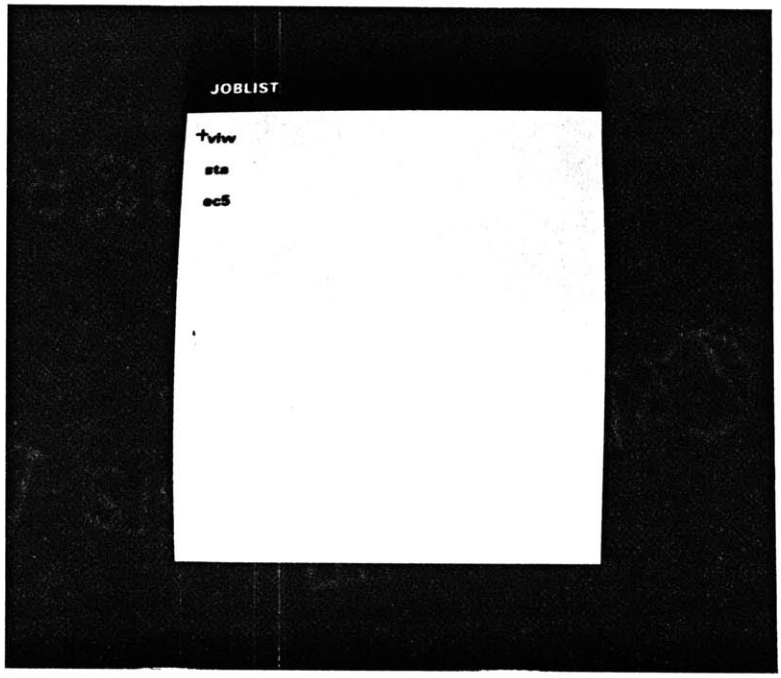
Example Session

CONFORM was designed to assist graphic designers with the process of organizing and integrating text and graphics. It can also be used by anyone who needs to organize verbal and/or visual information for technical documents, proposals, annual reports, catalogs, books, or even on-line information systems.

A manuscript that is created by an author exists as an ascii text file in the computer. It may have been written using a word processor, and then transferred via a telecommunications network, or loaded from a floppy disk.

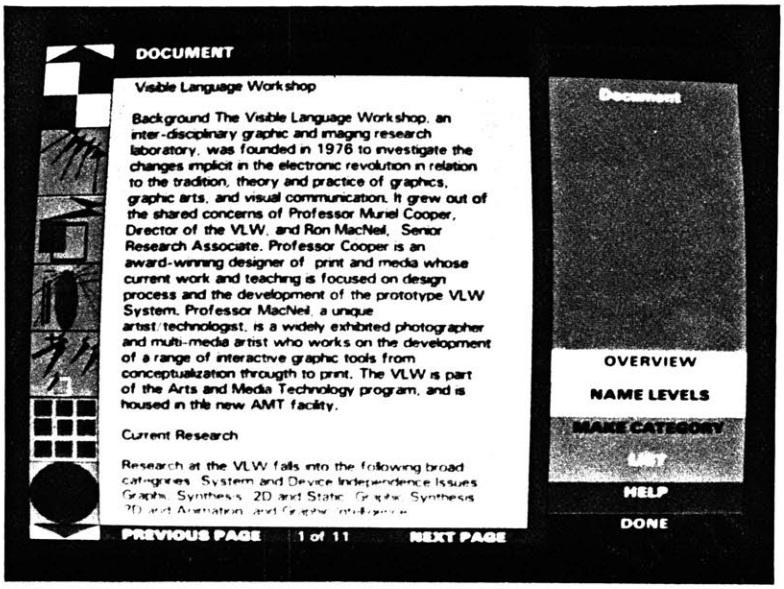
Images may come from many sources and range from line art on paper to picture description instructions for producing CAD drawings, or color reflective hardcopy to a bitmapped image of variable spatial and color resolution. Source images that do not reside in computer memory can be digitized and placed in the system through a frame grabber.

CONFORM assumes that the text exists as an ascii text file, and that images have been scanned into the computer. Following is an illustration of a work session using CONFORM.

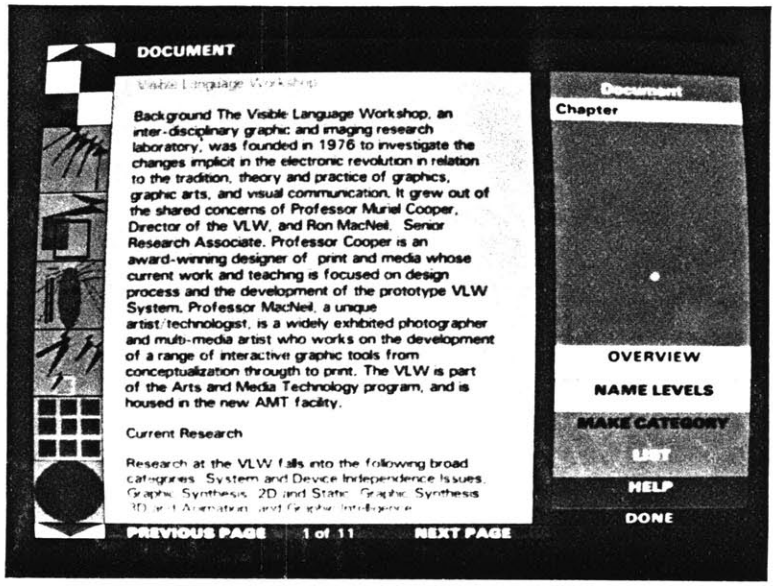


The user begins by selecting the job from the list that they want to work on by pressing the middle mouse button.

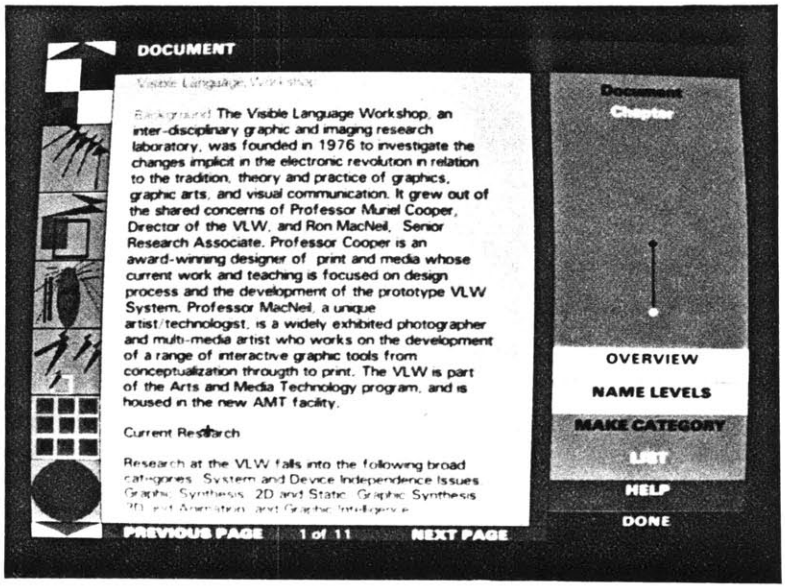
The associated job files are loaded into the system, and the text and images are displayed on the screen. The user can begin chunking the text and linking images to it.

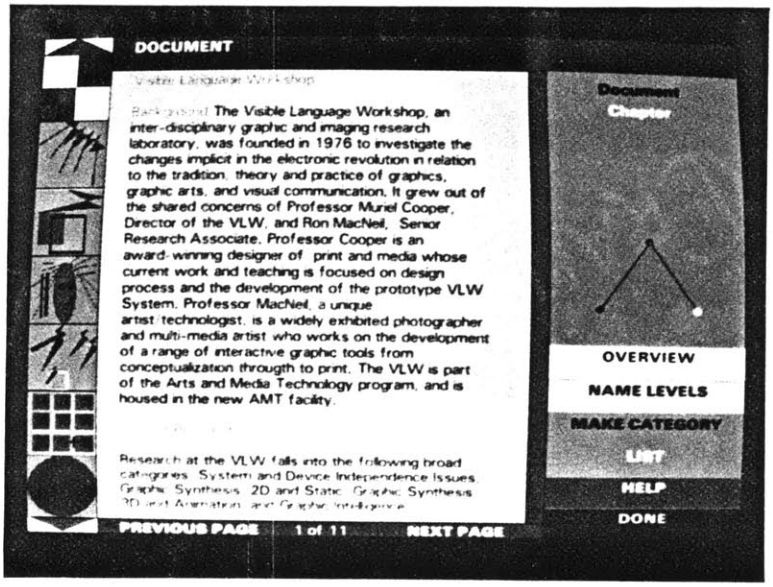


The first text string selected is the top node of the tree, and it is displayed in the tree/menu area as a white circle. Since there can only be one top node, the tree level is automatically incremented, and the default label displayed. The user can define their own label by typing it in at the keyboard, or use press the middle mouse button to use the default.

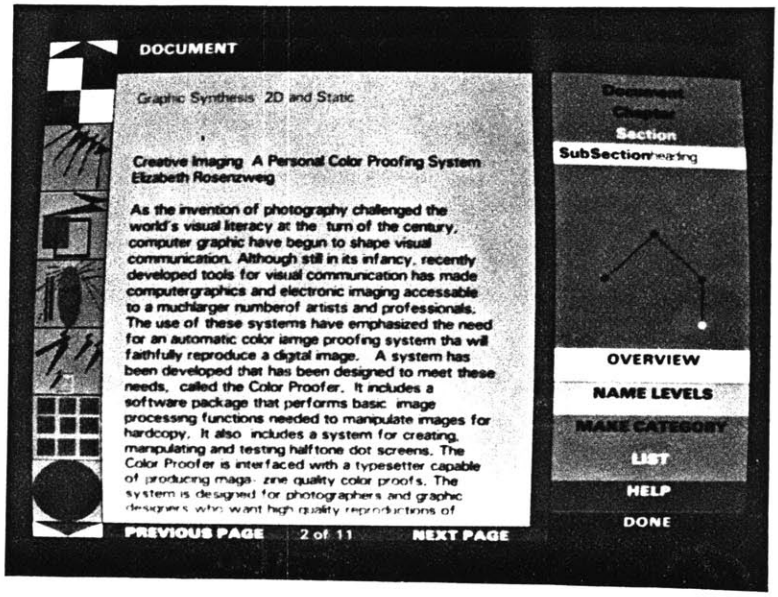


As strings are selected additional nodes are added to the tree at the same level.

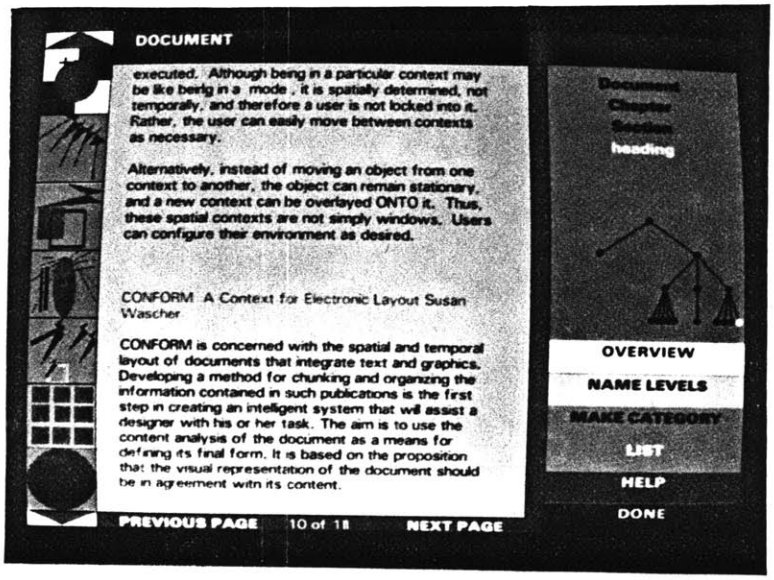


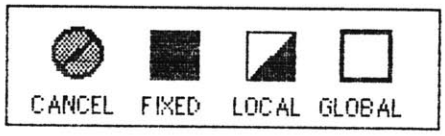


Levels are changed by pressing either the right or left mouse button. The left button (PREVIOUS) moves up the tree, and the right one (NEXT) moves down it. Here a level is being selected for the first time and its default label appears above the tree. To change it, the user simply types in their own label at the keyboard, hitting the return key to terminate the input.

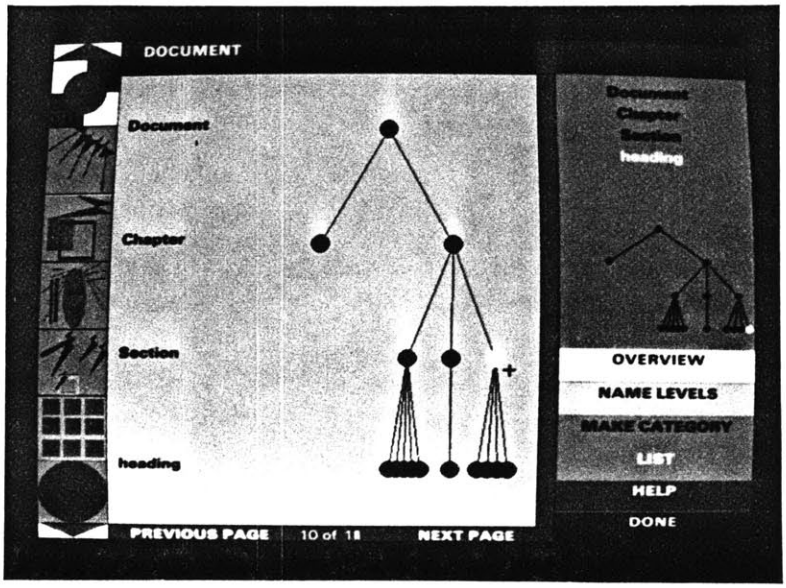


Images are selected by pointing to them and pressing the SELECT button on the mouse. A red circle appears over the selection to show that it has been selected. The cursor is automatically repositioned in the page area for the user to select the string it is to be linked with. Once selected, a menu pops up which allows the user to select the flexibility of the link or cancel the selection entirely.

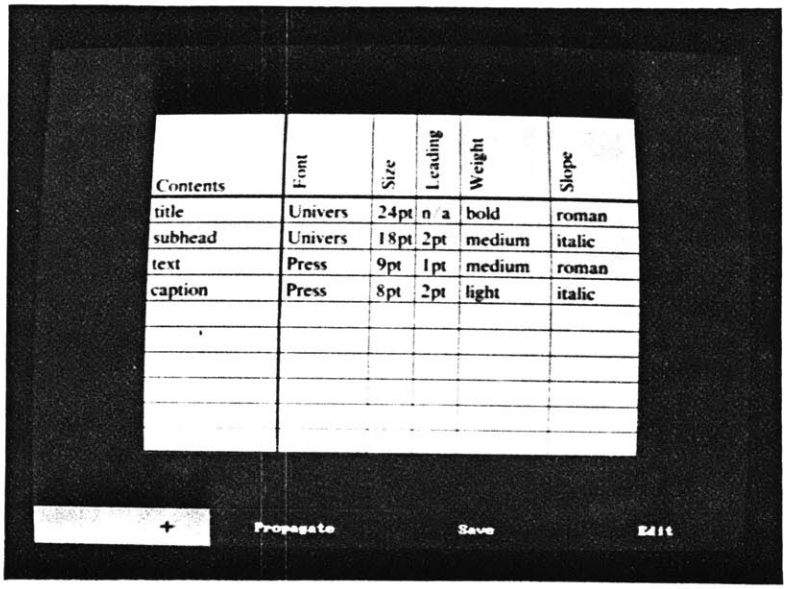




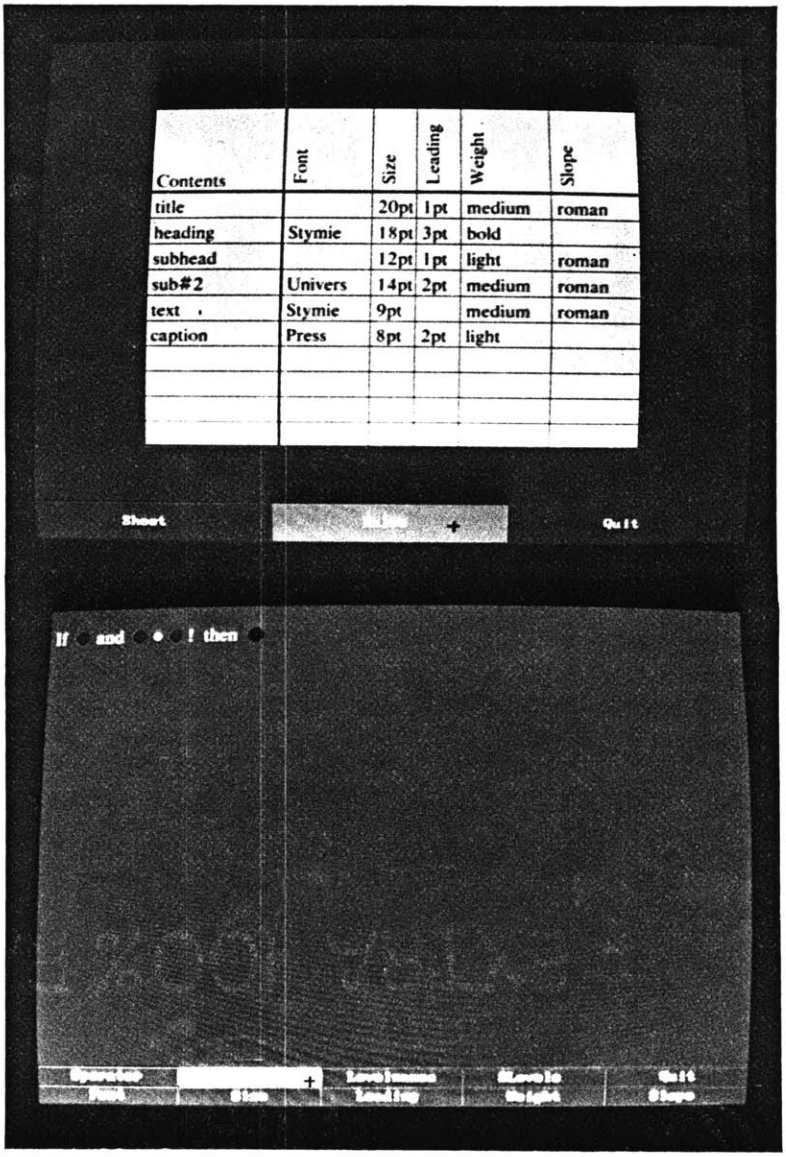
Selecting the menu option OVERVIEW causes a larger version of the tree to be displayed in the page area. This provides the user with an abstract overview of the document from which they can select nodes and see their associated text strings. It is also used for accessing different parts of the text for display in either the page area or PREVIEW.



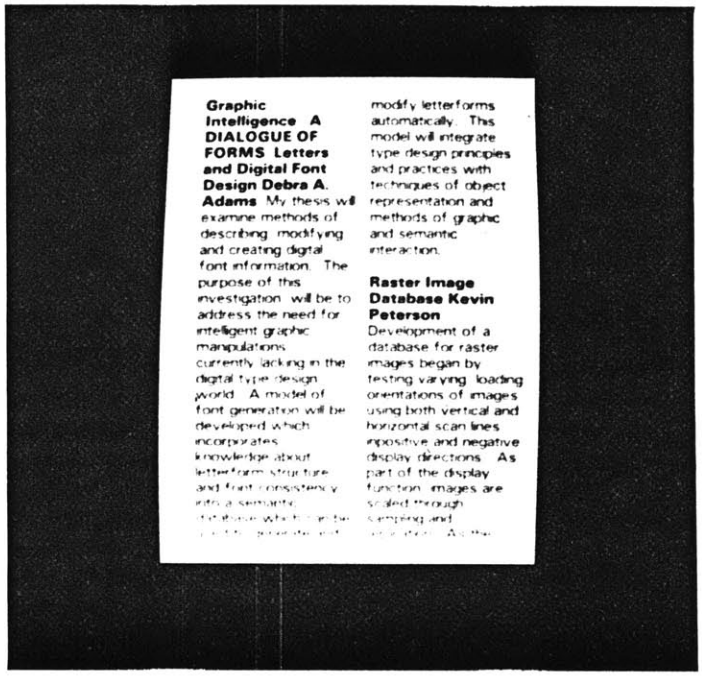
To preview a document in page form, the user selects DONE from the menu options. A typographic style sheet that has been stored previously can be loaded and used to for setting the typographic parameters of the page.



A blank typographic style sheet can also be loaded. The user specifies their own parameters, or has them generated based on information from the tree data structure and rules supplied by the system or rules defined by the user.



PREVIEW shows the text in a preselected format with typographic parameters accessed on the fly from the typographic style sheet as the text is flowed into the columns.



Graphic Intelligence A DIALOGUE OF FORMS Letters and Digital Font Design Debra A. Adams My thesis will examine methods of describing, modifying, and creating digital font information. The purpose of this investigation will be to address the need for intelligent graphic manipulations currently lacking in the digital type design world. A model of font generation will be developed which incorporates knowledge about letterform structure and font consistency into a semantic

modify letterforms automatically. This model will integrate type design principles and practices with techniques of object representation and methods of graphic and semantic interaction.

Raster Image Database Kevin Peterson Development of a database for raster images began by testing varying loading orientations of images using both vertical and horizontal scan lines, inpositive and negative display directions. As part of the display function, images are scaled through sampling and

Conclusion

CONFORM successfully supports the user in defining the structure of a document, and relating images to the text. Document segments are organized into a tree data structure to facilitate formatting based on their relationships. A typographic style sheet has been created that uses information from the tree structure to determine typographic parameters. PREVIEW demonstrates how the tree structure and embedded codes are used with a spatial representation and typographic style sheet to instantiate a page.

The structured document approach with embedded category codes has been successful in developing and linking with representations of form. This is evidenced in the work done on the typographic style sheet. Information from a tree structure is used with rules to propogate the values of the sheet. The graphic representation of the tree, however, requires additional searching for a good solution.

Editing of the document is not currently supported, as the main focus of this work was on developing structures and representations that could be used in creating a designer's workstation with the ability to provide assistance to the user. Basic editing functions, however, have been written, and are used in naming the levels of the structure. They could provide the basis for developing a color text editor.

The scope of this work has broadened since its inception. Primarily developed as a software package for print publishing systems, it also seems to be an appropriate approach to designing frames of information for electronic delivery systems. It is evident that creating separate representations for content and form not only allows flexibility in formatting a document, but accessing and viewing information as well. Now the reader of a text can view it in their own style.

Future Work

Following is an outline for futher work based on what has been established with CONFORM.

STRUCTURE REPRESENTATIONS

Investigation of various graphic representations based on the defined structure of the document to aid the user in navigating the information.

LINKING TO OTHER REPRESENTATIONS

Other visual representations to generate a page or layout need to be considered. Besides the spatial representation, various charting methods need to be developed to show the temporal layout of the document. Information from the tree could be used to show relative size of document segments, and distribution of images within the text. The distribution could automatically change using the flexibility value of the image link.

ENVIRONMENT

User control of the arrangement of the space and its contrast should be supported. Studies need to be consulted in determining the best range of font sizes and background/foreground colors to avoid or minimize eye strain. More refined kerning tables and filtering algorithms need to be looked at for the antialiased text.

IMAGES

Much work needs to be done to display multiple images, each with their own unique color tables, in the reference space, and on the page in the instantiated version of the document. Images used to illustrate the use of CONFORM were 150x150 pixels or smaller, all with the same 40 value colortable. Since images are only shown in CONFORM as scaled versions, the color resolution of the image is already reduced. However, a method for quantizing the color is needed to limit the values to 60 vlt slots, and relating them to the appropriate pixel. The remaining 196 slots are used for antialiased text, and in creating the environment.

ALTERNATIVE METHOD FOR CHUNKING

Development of a method for chunking using a document outline. If the outline exists, it could be used to flag the text file through string pattern matching similiar to a global search and replace function in a text editor. However, the user will need to resolve more than one instance of the string being matched.

RULES AND GUIDELINES

Extension of the current work to develop a set of style sheets to support graphic standards for a corporation. If several typographic style sheets are used to represent the standard, design the system to pick the appropriate style sheet based on the levels of information in a particular document.

IMAGE DATABASE SEARCH

Use the image list generated from CONFORM as instructions to search an image database for visual reference material or images to be used in the document.

Appendix

Tnodes "Tnodes.c" contains the text node structure and functions to access it. The structure should always be accessed via these functions, rather then directly. This will allow for later changes, if necessary.

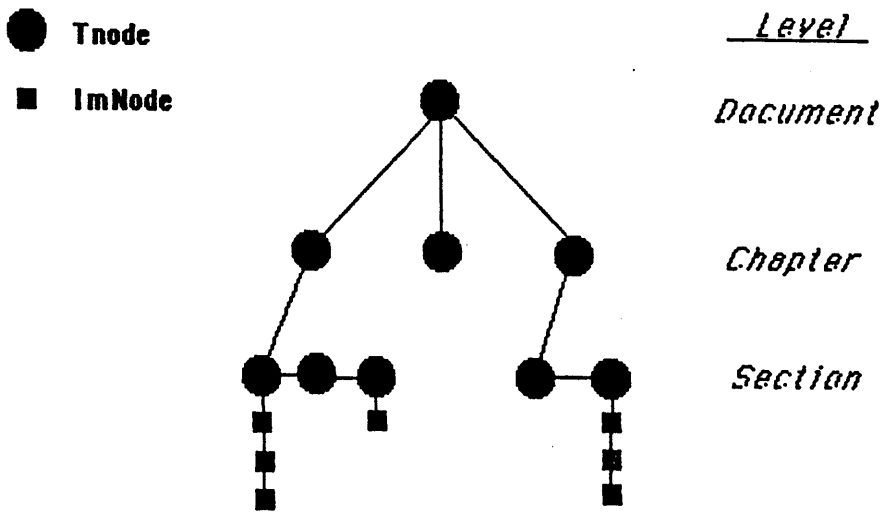
Each text node (**Tnode**) consists of these fundamental parts:

1	filespec	file specification of text
2	offset	offset of text string in **filespec** n-number of offset bytes from beginning of file
3	ImageList	list of images associated with this node (actually, pointer to first **ImNode**, if any)
4	parent	pointer to parent of this node
5	child	pointer to first child of this node
6	next	pointer to next (sibling) node
7	prev	pointer to previous (sibling) node
8	x, y	screen coordinates computed in **CalcCoords()** (accessed when drawing the tree)

Each image node (**ImNode**) consists of:

1	imspec	file specification of image
2	imflex	number indicating the flexibility of the image link 0 - rigid 1 - local 2 - global
3	imnext	pointer to next **ImNode** in list

Tnodes are maintained in a tree structure, while **ImNodes** are maintained in a linked list.




```
struct Tnode *tp, *parent, *child, *next, *prev, *tree;
struct ImNode *ip, *imnext;
int level, imflex, x, y, r, lc, cc;
long offset;
char *filespec, *filename, *imspec;
pmd *pm;
```

Type	Function	Description
int	level(tp)	returns level number (0 is document level)
char *	filespec(tp)	returns filespec of text
long	offset(tp)	returns offset of text from start of file
int	x(tp)	returns screen x-coordinate of node
int	y(tp)	returns screen y-coordinate of node
int	children(tp)	returns number of children (0 if none)
int	NumImages(tp)	returns number of images in ImageList (0 if none)
struct ImNode *	ImageList(tp)	return pointer to first image (NULL if none)
struct Tnode *	parent(tp)	returns pointer to parent node (NULL if none)
struct Tnode *	child(tp)	returns pointer to first child node (NULL if none)
struct Tnode *	next(tp)	returns pointer to next node with same parent (NULL if none)
struct Tnode *	prev(tp)	returns pointer to previous node with same parent (NULL if none)
int	set_filespec(tp,filespec)	sets filespec to specified value
int	set_offset(tp,offset)	sets offset to specified value
int	set_x(tp,x)	sets screen x-ccordinate to specified value
int	set_y(tp,y)	sets screen y-ccordinate to specified value
int	set_ImageList(tp,ip)	sets pointer to first image to specified value
int	set_parent(tp,parent)	sets parent to specified value
int	set_child(tp,child)	sets first child to specified value
int	set_next(tp,next)	sets next to specified value
int	set_prev(tp,prev)	sets prev to specified value
char *	imspec(ip)	returns filespec of image
int	imflex(ip)	returns flexibility of image
struct ImNode *	imnext(ip)	returns pointer to next image in list
int	set_imspec(ip,imspec)	sets image filespec to specified value
int	set_imflex(ip,imflex)	sets image flexibility to specified value
int	set_imnext(ip,imnext)	sets next image pointer to specified value
struct Tnode *	NewTnode()	returns pointer to new null-initialized Tnode (NULL if out of memory)
struct ImNode *	NewImNode()	returns pointer to new null-initialized ImNode (NULL if out of memory)
int	Save(tree,filename)	saves tree into ascii file named filename (NULL if error opening file)
struct Tnode *	Load(filename)	returns pointer to tree loaded from filename (NULL if error loading file)
int	PrintTnode(tp)	prints information about Tnode to stdout
int	NumLevels(tree)	returns number of levels (including 0) in tree
int	NumNodes(tree,level)	returns number of nodes on level of tree
int	CalcCoords(tree,wide,high)	calculates and sets screen coordinates for all nodes of tree (will fit in area wide x high)
int	DrawTree(tree,pm,r,cc,lc)	draws tree in pixelmap pm with solid circles of radius r and color cc connected by lines of color lc
struct Tnode *	NearNode(tree,x,y)	returns node in tree nearest to screen point(x,y)

MAIN PROGRAM Main program is in file "cform.c". Its modules include:

main()

Control code for CONFORM.

pgprint(npg, tpgs)

Print current page number as **npg** of **tpgs** at bottom page scroll bar (e.g. "1 of 11").

setscreen()

Setup CONFORM screen environment.

OverView()

Overview menu selection. Returns pointer to chosen tree node.

WriteLabels(lvl)

Write level labels on screen, highlighting label for given **lvl**.

Data Structures

Two data structures are used to store information about the text and the region it is written to.

struct text_region

```
{   pmd *area;
    int wd;
    int ht;
    int lmargin;
    int rmargin;
    int tmargin;
    int bmargin;
    int color;
}
```

struct text

```
{   char fname[16];
    long linebr[512];
    int maxlns;
    int bsize;
    char *buf;
}
```

Text Functions

"prtext.c" contains the following functions to print text to the display:

y2printp(rptr, font, color, tptr, line_num)
print page of text in specified **font** and **color** in the region **rptr** points to starting at **line_num** in text buffer (**tptr**)

ygetline(tptr, line_num, line)
reads characters in text buffer pointed to by **tptr** starting at **line_num** and returns them in **line[]**

ygetstring(tptr, line_num, lmargin, x1, x2, string, index1, index2)
returns **string** within a screen line, given **line_num, x1**, and **x2**, in string. Also returns start and end indices within buffer.

getpgln(y, maxdepth, ht, maxh)
returns screen line number

getword(tptr, pt1, word, pt2, code)

returns **word** from buffer and the indices for its beginning and end, as well as the embedded code or zero.

printpp(rptr, font, color, tptr, line_num)

prints page of text in given **font** and **color** starting at **line_num** in buffer using embedded codes

setparms(code)

set **font** and **color** parameters according to embedded **code**

AddTnode(fn, pos)

Add new **Tnode** to global Tree, connected to global TP, according to gloabal DOCLEVEL

getflex()

Get and return image flexibility number.

AddImNode(imfn, flex)

Add new **ImNode** to global TREE, connected to global TP. Set filespec and flexibility to **imfn** and **flex**.

InsertCodes(tptr, pos1, pos2, ln1, ln2, code)

Add **code** and BODY characters to buffer surrounding text between **pos1** and **pos2**, on lines **ln1** and **ln2**.

SelectString(rptr, font, x1, x2, y1, y2, tptr, firstln, nextln, color, ppos1, ppos2, pln1, pln2)

Highlight text selected by **(x1, y1)** and **(x2, y2)** in given **font** and **color**, then return starting and ending positions and lines in **ppos1, ppos2, pln1**, and **pln2**.

"gettext.c" contains text functions for reading the text file into the buffer and calculating the line breaks for the display.

gettext(tptr)

Loads file identified by **tpr->fname** into buffer and returns the number of characters in the file.

y2calcln(tptr, font, maxlen)

Calculates the line breaks given **font** and **maxlen**, and stores them as buffer offsets in **tptr->linebr[]**.

"cfm_text.c" contains functions to retrieve the joblist and the filename associated with a job.

get_jblist()

Gets the list of jobs from "joblist.h", puts them in **jbname[]**, and returns the number of jobs (**jbnum**).

get_job(jbnum, maxh)

Return the number of the job selected by the user.

load_tnames(job)

Given name of **job**, gets list of text files from "text.h" and puts them in **tname[]**. Returns number of text files.

PREVIEW Functions for previewing a document are located in "preview.c".

preview(tp, tptr)

Print formatted page of text from **Tnode** pointer **tp** and text pointer **tptr**.

yprint(rptr, font, tptr, coffset, noffset)

Print word at current offset (**coffset**), returning the new offset (**noffset**).

set_typoparms(code, color, ln)

Set typographic parameters for given level **code**, in given **color**.

set_pscreen()

Setup screen for preview function (page and regions).

Functions for defining tree labels are located in "labels.c".

GetNewLabel(lvl)

Gets a new label for given level (**lvl**), and puts the result into global **label[lvl]**. Defaults to **def_label[lvl]**.

SaveLabels(fn)

Save **label[]** into file named by **fn**.

LoadLabels(fn)

Load **label[]** from file named by **fn**.

Image Functions

"cfm_img.c" contains the following functions for loading and scaling images:

previmg(num)

Get previous image - scroll image area down.

nextimg(num)

Get next image - scroll image area up.

loadimg(job)

Load, scale, and draw all images for **job.**
Calls **draw_icons.**

draw_icons(name, destpmd)

Draw scaled version of image (**name**) in the destination pixel map (**destpmd**).

get_img(name, img, src_img)

Gets the image identified by **name**, sets **img** dimensions, and places image in source pmd (**src_img**).

SelectImage(ImNum)

Select (or deselect, if already selected) image (**ImNum**) on screen with red box. Automatically deselects old **cur_img** and sets cur_img to **imnum**.

Scaling function used to draw the images in the destpmd is found in "scale.c".

Miscellaneous

Other files containing functions used in CONFORM:

"mouse.c" contains the mouse control routines.

"tnodes.c" contains the text node and tree manipulation routines.

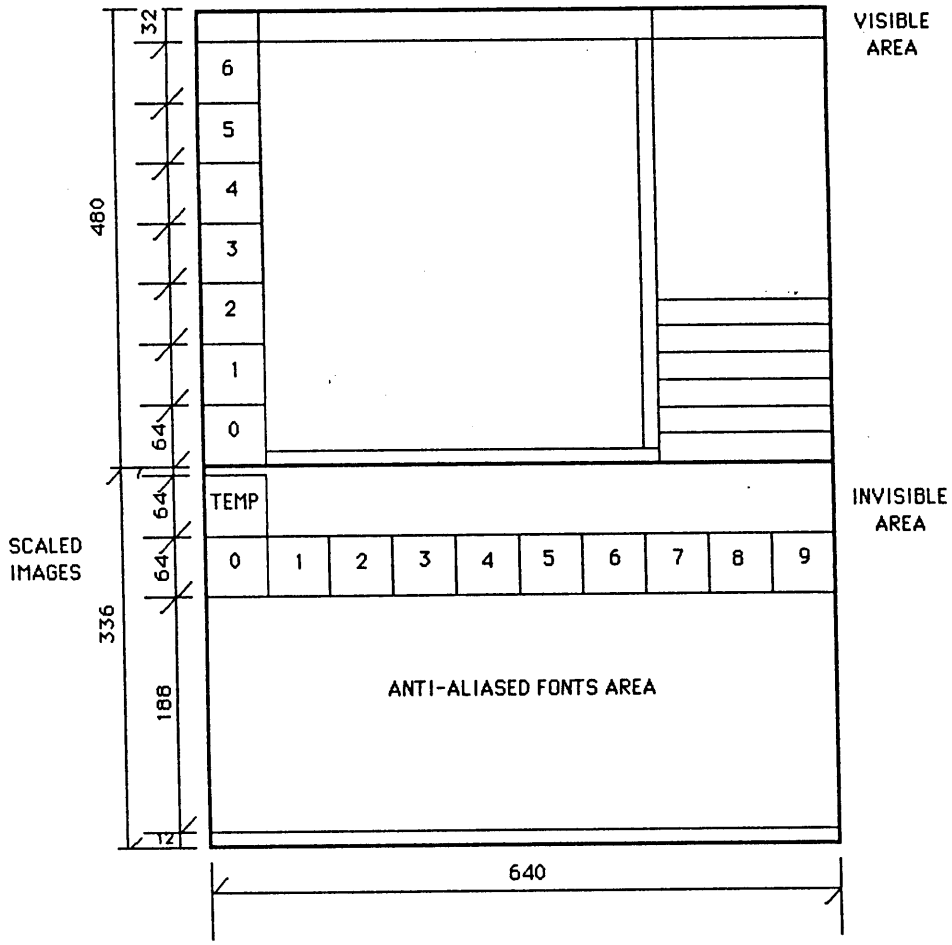
"cursor3.c" contains mouse and YODA cursor control routines.

"setvlt.c" creates the environment for CONFORM.

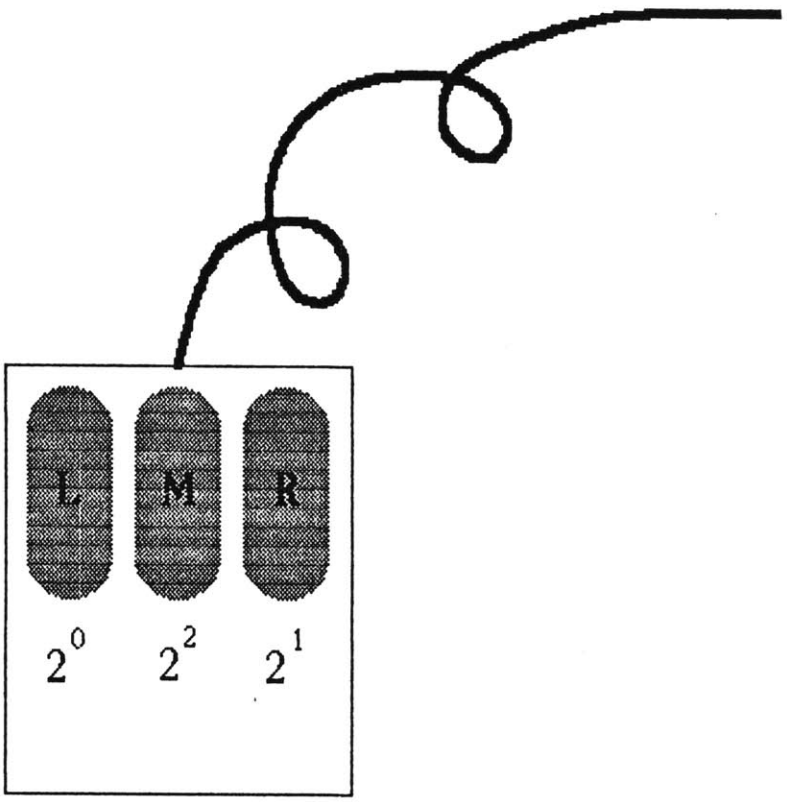
CONFORM Colortable

SLOT	RED	GREEN	BLUE	NAME
0	0	0	0	BLACK
15	255	255	255	WHITE
16	63	63	63	DK GREY
31	255	255	255	WHITE
32	127	127	127	MD GREY
47	255	255	255	WHITE
48	127	127	127	MD GREY
63	0	0	0	BLACK
64	191	191	191	LT GREY
79	0	0	0	BLACK
80	255	255	255	WHITE
95	0	0	0	BLACK
96	191	191	191	LT GREY
111	255	0	0	RED
112	191	191	191	LT GREY
127	207	0	0	LT RED
128	191	191	191	LT GREY
143	159	0	0	MD RED
144	191	191	191	LT GREY
159	111	0	0	MD DK RED
160	191	191	191	LT GREY
175	63	0	0	DK RED

YODA Frame buffer The following diagram illustrates the major areas of CONFORM's task environment, and their pixel map location in relation to the frame buffer.



MSMOUSE



2^2	2^1	2^0		
0	0	0	= 0	None Depressed
0	0	1	= 1	Left Button Depressed
0	1	0	= 2	Right Button Depressed
1	0	0	= 4	Middle Button Depressed
0	1	1	= 3	Left & Right Depressed
1	0	1	= 5	Left & Middle Depressed
1	1	0	= 6	Right & Middle Depressed
1	1	1	= 7	All Depressed

Anderson, John — Cognitive Psychology and Its Implications
Freeman, San Francisco, 1980

Arnheim, Rudolf — Art and Visual Perception
University of California Press, Berkeley, 1954.

Association of American Publishers — Project Plan and RFP for the Development of Publishing Industry Standards and Author Guidlines on Electronic Manuscript Preparation
AAP New Technology Committe,
Electronic Publishing Subcommittee,
May 6, 1983.

Beach, R. and, Stone, M. — "Graphical Style: Towards High Quality Illustrations"
Computer Graphics, Vol. 17, No. 3, 1983.

Bolt, Richard — Spatial Data-Management
MIT, 1979.

Burt, Cyril — A Psychological Study of Typography
Macmillan, London, 1959

Caplan, Ralph — By Design - Why There are no Locks on the Bathroom Doors in the Hotel Louis XIV and Other Object Lessons
St. Martin's Press, NY, 1982.

Cherry, Colin — On Human Communication
MIT Press, 1978.

Craig, James — Designing with Type
Watsun-Guptill, NY, 1980

Davis, Randall "Expert Systems: Where are We?
And Where do We go from Here?"
MIT, Artificial Intelligence Laboratory,
AI Memo no. 665, June 1982.

Dondis, Donid A. A Primer of Visual Leteracy
MIT Press, 1973.

Doubleday The Doubleday Dictionary
Doubleday and Co., NY, 1975.

Feiner, Steven et al. "An Integrated System for Creating and
Presenting Complex Computer-Based Documents"
Computer Graphics, vol.15 no.3, August 1981.

Foley, JD
Van Dam, A Fundamentals of Interactive Computer Graphics
Addison-Wesley, 1982.

Gerstner, Karl Compendium for Literates
Weber AG., Heiden, 1974.

Glenn, Bernice "Alphabet and Text in Presentation Graphics"
Computer Graphics World, January 1984.

Hamilton, Edward A. Graphic Design for the Computer Age
Van Nostrand Reinhold, NY, 1970.

Hurlburt, Allen Publication Design
Van Nostrand Reinhold, NY, 1976.

IBM YODA 0.2 Programmers Guide, Version 2.0
IBM Yorktown Heights, NY, February 1985.

Knuth, Donald — The TEXbook
Addison-Wesley Publishing Company, 1984.

Lipkie, DE et al. — "Star Graphics: an Object Oriented Implementation"
Computer Graphics, Vol. 10, No. 3, 1982.

Miller, George — The Psychology of Communication: Seven Essays
Basic Books, NY, 1978.

Naisbitt, John — Megatrends
Warner, NY, 1982.

Olivetti — "Design Process"
Design Process Olivetti: 1908-1978
Ing.C. Olivetti and C., 1979.

Rand, Paul — Thoughts on Design
Studio Vista Limited, Van Nostrand Reinhold, London, 1970.

Ruder, Emil — Typography: A Manual of Design
Hastings House, Publishers, Inc., New York, 1981

Schank, Roger C. — The Cognitive Computer
Addison-Wesley, Reading, MA, 1984.

SCITEX — Vista System Description
Scitex Corporation Ltd, 1983,
Publication no. 137H3A1.

Scitex Response 300
Scitex Corporation Ltd, 1984,
Publication no. 168G401.

Seybold, Jonathan — The Seybold Report on Publishing Sestems
Seybold Publications,
Vol. 14, No. 6, 1984; No. 9, 1985.

SIAD/STD — Computers in Visual Communication
SIAD/STD Typograppher's Computer Working
Group Symposium Proceedings, 1969.

Siebels, Susan — "Mnemonic Coding: From Word Processing Input
to Typesitting Output"
STA Workshop Abstract, Spring 1985.

Simon, Herbert — The Sciences of the Artificial
MIT Press, 1969.

Spencer, Herbert — Pioneers of Modern Typography
MIT Press, 1969.

Stankowski, Anton — Visual Presentation of Invisible Processes
Hastings House, NY, 1967.

Tschichold, Jan — Asymmetic Typography
Faber and Faber, London, 1967.

Tufte, Edward — The Visual Display of Quantitative Information
Graphics Press, Cheshire, Conn., 1983,

White, Jan V. — Editing by Design
Bowker, NY, 1974.

Wilson, Adrian — The Design of Books
Reinhold, NY, 1967.

Zorkoczy, Peter — Information Technology: An Introduction
Van Nostrand Reinhold, NY, 1982.

Acknowlegements CONFORM was supported in part by Hell GmbH. I'd like to credit David Lively for his excellent work in implementing the tree structure, and the functions to access and manipulate it. Also, Tom Thomas did a remarkable job in developing an expert system for generating a typographic style sheet. I'm indebted to both these individuals for their significant contributions to this work, and enjoyed working with them.

This work is dedicated to all those who believed in me, even when I didn't believe in myself. But, especially to my Dad who taught me to think for myself, and my Mother who showed me that women can be mechanically adept. Special thanks goes to Prem, without whose help and support this document may not exist.

Thank you all.