MAN/MACHINE INTERACTION IN COMPUTER ASSISTED PAGE LAYOUT

by

Paul Byers Trevithick

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF BACHELOR OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May, 1981

Signature of Author [illegible]
Department of Electrical Engineering
May, 1981

Certified by [illegible]
Thesis Supervisor

Accepted by [illegible]
Chairperson, Departmental Committee on Theses

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

This thesis examines the man/machine interface of a prototypical text and image handling system for the production of printed material. It also describes software written by the author on the human interface part of the system. Where necessary or applicable, other hardware and software parts of the system will be described.

The "text and image handling system" under development at the Visual Language Workshop is defined this broadly because it attempts among other things to combine the individual functions of contemporary "word processors," page layout systems, typesetters, graphic design systems, editors, and text formatters. It will ultimately be capable of accepting high resolution input imagery, high quality type, text input, and to manipulate and format pages of material which are ultimately output by high quality scanners.

The hardware is shown in Figure 1. It consists of a 32 bit "super-mini" computer, a digital tablet, a 300 Mbyte disk, a high resolution CCD scanner, a frame buffer, and a high resolution video monitor. The tablet and video monitor comprise the user's workstation. For the remainder of this thesis, two color monitors will be assumed; however, at present, one monitor switches between two images.

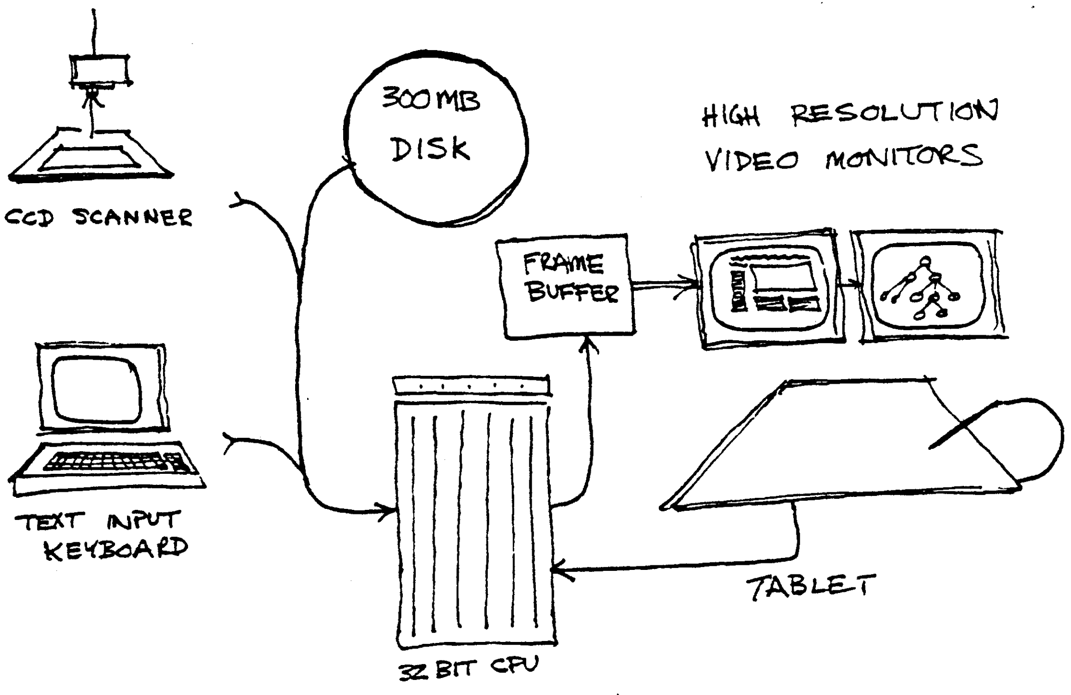


Figure 1.

The human interface on this system has been designed with three broad criteria in mind. First, immediate "pre-visualization" is provided to the user so that an instant evaluation can be made of the action just taken or function performed. Second, whenever possible and applicable, graphical methods of input have been substituted for keystrokes. And third, the control methodology and system documentation should be based on an epistemological structure of the process of text and image manipulation, which allows the creation of a self-consistent, straightforward, personalized and self-documenting user interface.

The need for "pre-visualization" is very evident. The turnaround time for even the fastest typesetting systems is weveral minutes. A mistake in such a system must propagate through to the final hardcopy before it can be detected.. Several entire cycles throught such a system are inevitable. These systems can be though of as "batch" systems since they operate in entire cycles and not incrementally. In the same way that batch computer processing has become superceded by interactive processing, it is hoped that batch typesetting will be replaced with interactive typesetting. How the previsualization is generated and displayed is discussed in Chapter One.

The man/machine interface on even the most expensive and state-of-the-art systems are designed to be used by skilled users who have assimilated the contents of the inevitably thick "User's Manual" for the system. The actual interface protocols themselves have very little to do with the page layout or typographical process much less the design process. In Chapter Two a methodology of interaction is presented which is based on a knowledge structure of the processes which the system will be required to perform. This means that <u>how</u> the user controls the system is directly and intimately related to <u>what</u> it is he is trying to achieve. The interface is exceedingly easy to learn and offers the additional advantages of self-documentation based on the user's current level of understanding, and personalization.

CHAPTER ONE

Previsualization

Current typesetters work in a "batch" mode. They require the user to specify the position of the text to set, as well as several parameters like font, point-size, indentation and so on. The machine then runs hyphenation and justification as desired and begins to typeset the page using some form of hardcopy printer for output.

The process is blind. The user has no feedback of his actions until the film of paper page is produced. It is only then that he can perform any sort of evaluation. Unless the user counts the number of words in his text column, and does the arithmetic to predict how long the column will be, he has no way of knowing what the column will look like on the page. He is forced to do the design in his head and then to transmit his idea of the page to the machine through formatting commands which are often only specifiable numerically.

Better page layout systems (like the CAMEX system), do offer a schematization of the final page. A line drawing display provides the user with a form of previsualization in which the dimensions of the various parts of the page are shown. The user can interactively scale, crop and position the page elements useing a tablet for graphical input. The display completes the control loop by giving the user instant feedback on his every action. This is a tremendous advantage.

The system under development at the VLW offers an even better form of previsualization. It uses a color raster scan display for the previsualization display. This allows images to be rendered and displayed in a form which allows aesthetic decisions to be made on the best position, shape, etc, for each image on the page. The line drawing displays described above do not have this capability. Images are usually shown as rectangles with a line through them.

The system also attempts to combine many of the functions of previously discrete systems. It tries to offer the advantages of previsualization to areas which have never had its benefit. In order to make typographic decisions about the appearance of text on a page, it is necessary to have either an actual or a mental picture of how the type will look. At present type must be produced in some manner and then placed before the typographer who uses them as references in deciding how the page should look. The prototype system allows the typographer to see "exactly" what the final output will look like before it is produced. The word exactly is in quotation marks because what that word means is a compex subject. For the present let "exactly" be defined as follows:

> the equivalent image to what a high quality scanner of the same resolution would produce when given the eventual hardcopy as input.

The Hardware

Since it is impossible for the main computer to produce a video image 30 times per second (the standard screen refresh rate) that task is offloaded to a device called a frame buffer. This device stores the color of each dot, called a pixel, in a large memory. This memory is scanned out 30 times per second to the video screen while these locations are also available to be altered by the "host" computer.

In this way the computer can draw into the buffer the new or altered sections of the page image (frame) , while leaving the other memory locations, each of which corresponds to a dot on the screen, untouched. This greatly lowers the computational load on the computer while allowing full color images on the screen. Depending on the degree of color realism, the cost and complexity of the frame buffer can vary widely.

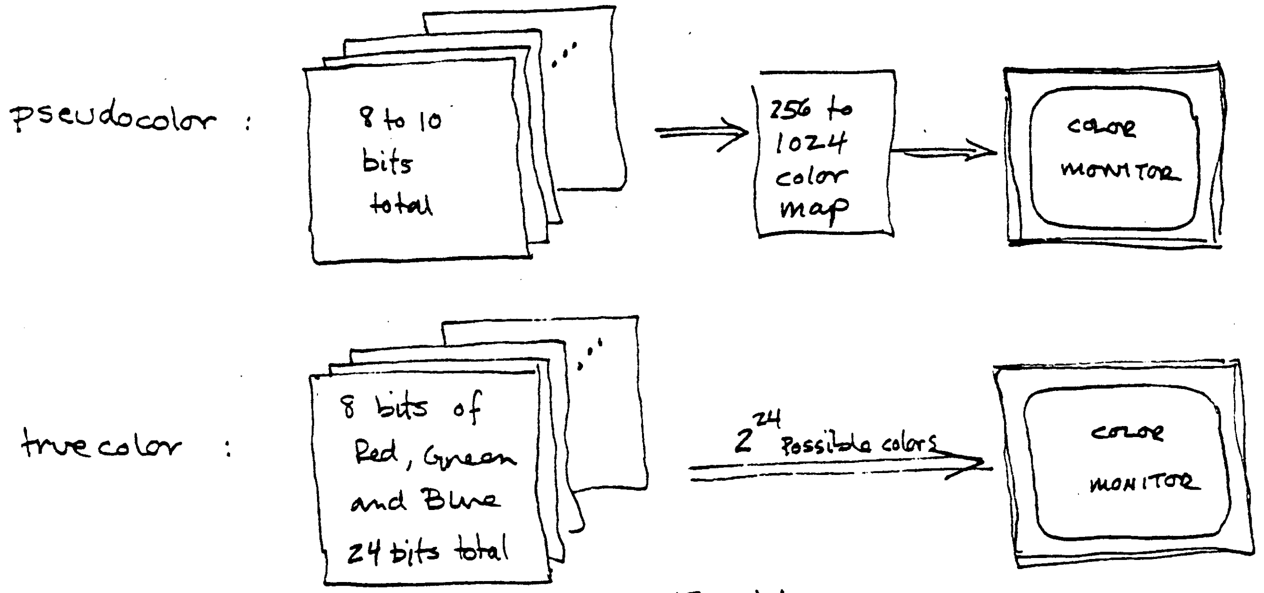


Fig 1.1

There are two ways to achieve color. The first is called pseudocolor. The frame buffer stores a number for each point on the screen. This number is mapped to an arbitrary color through a lookup table. This is the least expensive method, although only a palette of less than 1,024 colors are typically available at once.

The second method requires roughly three times the memory and thus adds considerably to the cost of the device. In this method three numbers are stored per pixel instead of one. The three numbers correspond directly to the amount of the three color primaries, red, green, and blue, at that position on the screen. The system used in this paper is of this latter type.

In many applications color is not required. However, it is easily shown that a choice of several brightnesses must be available at each pixel for the most accurate and legible page. A "one bit" (black or white only) display will produce the stairstepping problems so often seen in computer imagery.

The amount of memory storage required per pixel is related to how many grey levels or color levels desired. The number of memory bits required is related to the log base 2 of the number of grey levels desired. Thus one bit of storage per pixel will produce a two level black or white display; two bits will produce four grey levels; three bits will produce eight ; and so on.

Anti-aliased Display

There is a paradox involved in raster scan display. To a certain extent, a blurrier display can result in a display with more visual acuity. It is not necessarily true that a hard black and white edge will result in a more readable image. On a standard video monitor alternate lines are refreshed at a 30Hz rate which can cause scintillation in some cases unless grey levels are used to smear horizontal edges. This blurring or smearing of the image is technically called anti-aliasing, and is of major concern throughout the computer graphics field. Crow has written an excellent paper on this topic.[1]

Since every doubling of display resolution requires four times the memory storage and bandwidth of the display device, there are technological considerations which limit the maximum attainable resolution. 1000 lines is about the limit of current CRT display technology.

The precise number of grey levels required, and the exact legibility tradeoff, are somewhat subjective; however, when type becomes very small, on the order of a few pixels in height, grey tone capability makes the difference between an almost random black and white dot pattern, and legible type.

Without the use of grey levels, a computer rendering of an angled line looks like that of Figure 1.2a. Figure 1.2b shows the desired line with the raster grid super-imposed If each pixel is shaded according to the percentage overlap in that pixel of the line, the resulting line will be that shown in Figure 1.2c. The result, when viewed from the correct distance, yields a subjectively smoother line.

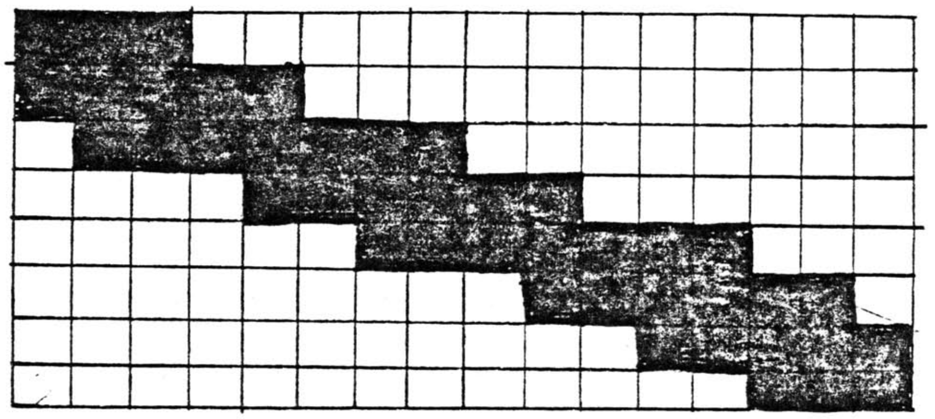


Fig. 1.2 (a)

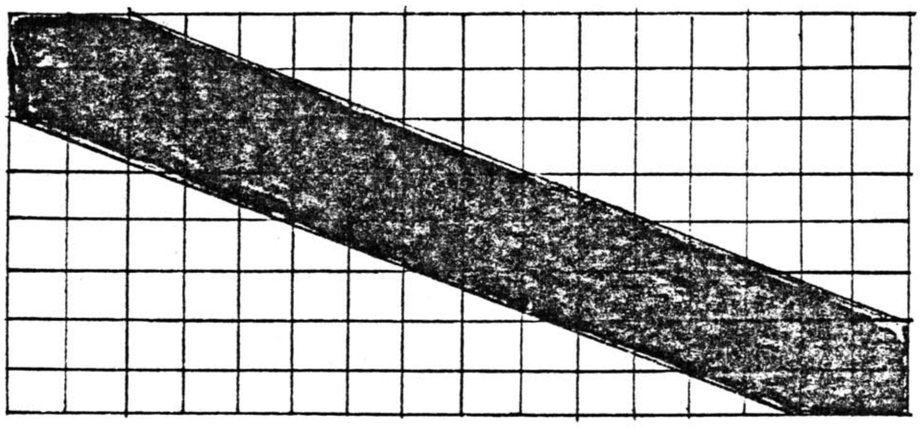


Fig. 1.2 (b)

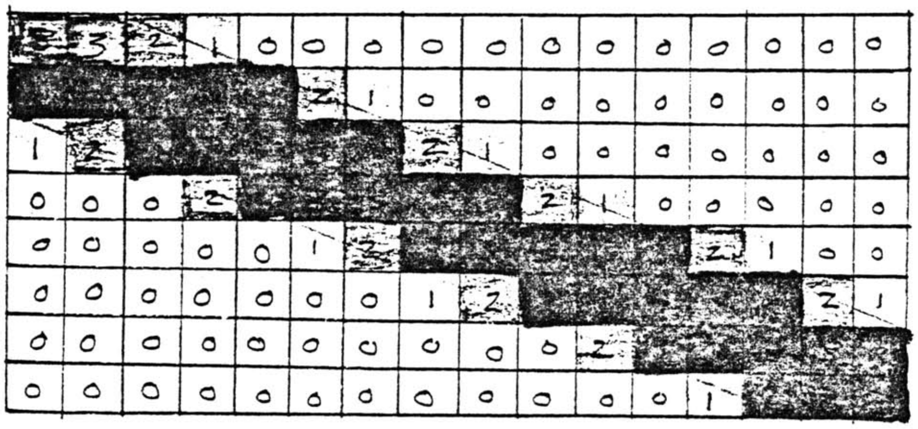


Fig. 1.2 (c)

Digital Fonts

A digital typesetting system requires an internal digital representation of the fonts it will be called upon to use. This internal data base can be generated in a number of ways. Commercial systems allow fonts to be loaded into the machines via diskettes or cassettes onto which have been copied digital fonts from a master file in some central system.

Since our system is being developed from scratch, and most commercial font packages are proprietary, we had to generate our own font data base. There are two broad ways of generating this data base. The first is to use a font generation system such as Metafont[2] or Ikarus[3]. Each of these systems allows the user to specify, numerically or graphically, parameters from which characters of the typeface are generated.

The second method involves digitizing type from existing collections of printed typefaces. Chris Schmandt[4], at the MIT Architecture Machine Group, has developed a digitization system which uses original printed masters and a high quality vidicon connected to a computer graphics system for input. The vidicon image of each character master was about 250 pixels in each direction. From this image a one bit black and white master was produced using a thresholding technique.

The Run-length Data Base

The choice of a font storage scheme was motivated by a desire for rapid display and for compact storage requirements. Since it was intended that both one bit black and white, and soft, or multi-grey, level type be displayable, the data base had to lend itself to either output method. All of the schemes considered involved digital storage since it was a necessity that the fonts be intimately connected with the computer system.

One possibility was to use a vector outline storage method. This is the most concise description of a character. (See Fig. 1.3a) for a given quality of result. The problem with this method of storage is that the output display routines are relatively complex and are therefore slow. The output

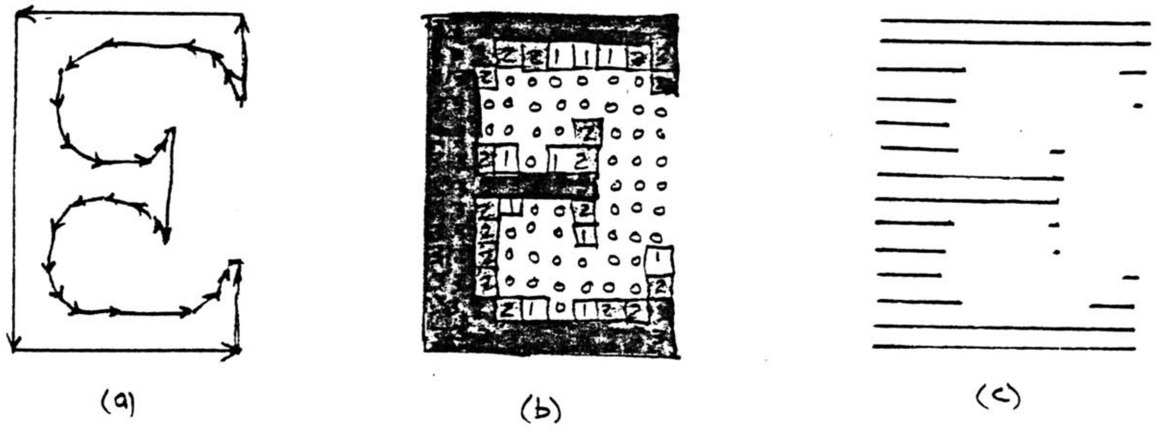

(a) (b) (c)

Fig 1.3

routine must slice the character at appropriate vertical intervals to scale it to the desired output size, and then find the intersection points of each slice with the vector outline. These points are then scaled in the X direction to yield the endpoints of a horizontal line to be drawn on the screen. Adding soft font capability to the output routine causes further complications.

The method shown in Fig. 3b has been extensively tested at the Workshop using eight bits per pixel storage, although four is definitely sufficient. The problems are threefold. First, the data base is very large. Second, the output routine is slow, and third, it is time-consuming to generate multiple font sizes. At the present, each font size is stored separately to speed up the display rate, which is still quite slow, causing enormous storage waste. The main advantages are that it is easy to implement, and gives soft fonts automatically since the pixel grey levels have been precomputed. A filtering program is used at present to shrink a master character to produce each smaller character. It also performs the blurring process referred to earlier. The output of such a program produces an array of grey levels which are then stored for each character.

The third method, shown in Fig. 3c, enjoys nearly the same compactness of storage as method (a), but makes it easy to display variable sized fonts rapidly. The masters of each character are stored as a list of horizontal runs. The letter E shown in Fig. 3c has a maximum of four runs. The first run is assumed to be white (background); the next black; the next white, and so on. By using this convention, the only information which must be stored is the horizontal x-coordinate (position) of where each run ends.

The master storage for the letter E, for example, consists of about 200 sets of horizontal runs where the maximum number of runs per set is four (for this font). Since the maximum resolution of the master is less than 256 (2^8) units in each dimension, the position information can be stored in one byte (8 bits). Thus the total storage for a capital E is about 800 Bytes. This means that the whole alphabet of one font can be stored within 64KBytes which is the maximum size of a data segment on the VLW 3220. Since it is now possible that the whole master font be resident in core at once, all core swapping is eliminated. When a new font is desired, it is brought from disk and written over the old one.

Font Digitization

To take advantage of the wealth of existing typeface collections, direct digitization of fonts was investigated. At first it was believed that a nuvicon camera could be used to scan information into the frame buffer; however, the noise levels were unacceptably high. Instead, a 2048 photodiode single line CCD scanner constructed by Professor MacNeil several years earlier was used. The intensity signal from every fourth diode was used to generate a 512 x 512 pixel image on the frame buffer by mechanically scanning the CCD array.

Despite the fact that the font originals were crisp black on white, the scanned in image contains greytones caused by particle overlap of a scanner sensor by a black and white edge. To extract the edges of each character, a grey level threshold was chosen which preserved the lightness or boldness of the type while generating hard black and white edges. The correct threshold was experimentally determined by using a program called "one bit." Only a fraction of the alphabet could be scanned in at once due to the need for high pixel resolution masters. The scanning time including set-up adjustments was about five minutes per section of the alphabet.

A program called "newfont" performs the actual font data base building, and its operation is briefly described. Using the tablet, the user stretches a box around the next letter to be scanned in. The program then scans through this box looking for black and white transitions on each horizontal row of pixels. Each time one is found, its position is recorded in a temporary run table. The temporary run table can handle up to 16 transitions per scan.

A search is then made through this temporary run table for the true left, bottom, and top of the character. The user is asked whether the lines at the top, left, and bottom of the character are the true perimeters of the character.

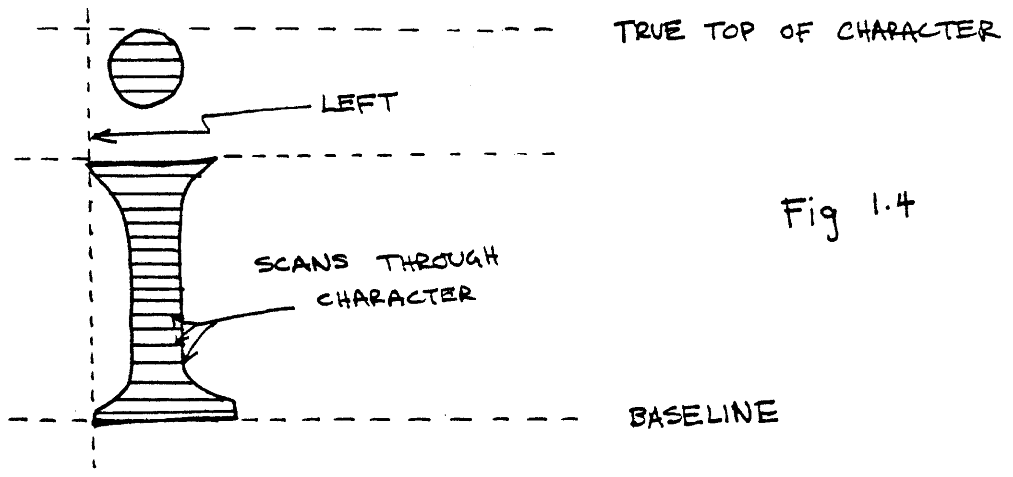


Fig 1.4

If so, the letter within those limits is displayed for the user to judge whether or not the letter has been scanned in correctly. The user is asked to point to the baseline position of the character.

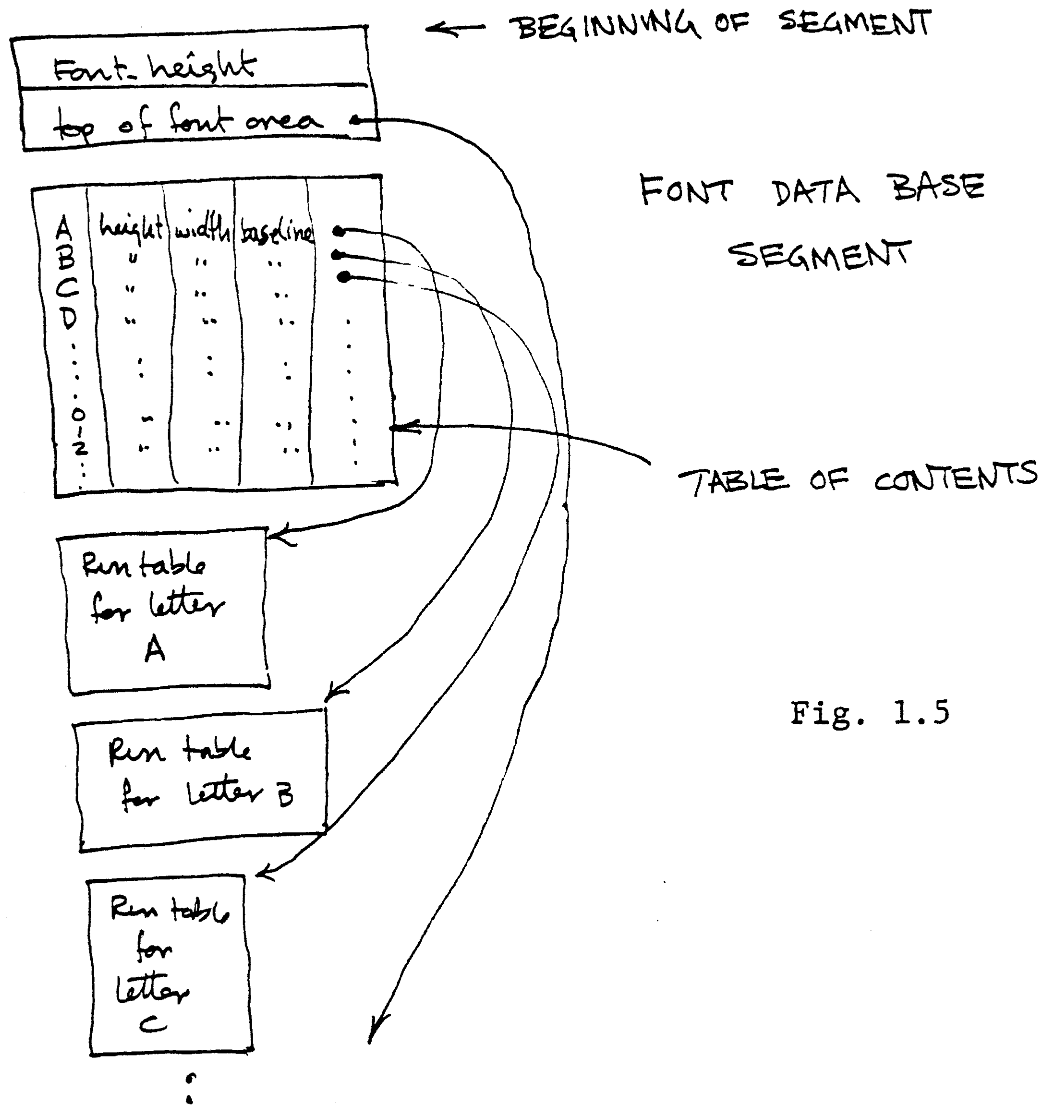
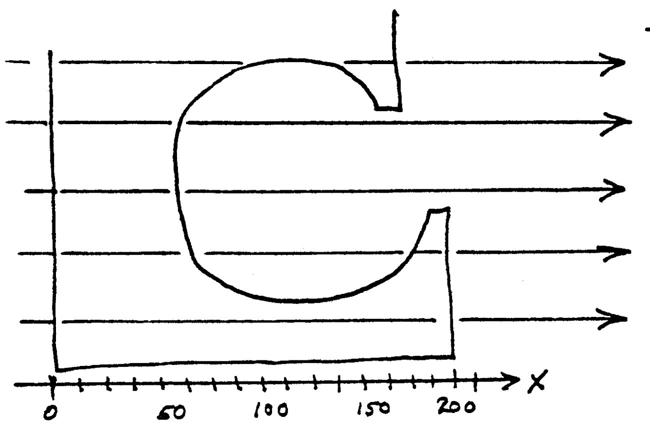


Fig. 1.5

As the scanning process proceeds, the maximum number of black and white transitions is recorded. This number is used as the width of the letter run table. For example, the width for the serif E would be four while the width for a simple sans-serif E would be two. By making variable width run tables, very efficient use of memory is achieved. Thus the width of table is determined by the horizontal complexity of the letter.

At the very beginning of the font data segment, the font-height (in scans) is stored. For each letter, number, and punctuation, there is an entry in a table of contents which is located near the beginning of the data segment. Four items are stored there per character; the height and width of the character's run table, the base line offset, and a pointer to the beginning of that character's run table. The run tables for each character are allocated and stacked one on top of the other in the remainder of the data segment.



1	80	130	175
1	51	0	0
1	50	0	0
1	55	160	200
1	200	0	0

Fig. 1.6

Letter scans and corresponding run table.

Font Display Techniques Using A Run-length Data Base

The display routine for variable size font in one bit is very simply. Since the font-height (in scans) was stored with the master, the masters can be thought of as being in their own absolute coordinate system. If the font size desired is specified in points, a conversion factor is used to scale that number to pixels. (On our system the conversion factor is 176/200 pixels/point.)

Using simple arithmetic the Y dimension can be scaled by subsampling,and the X dimension by multiplying the run numbers by a scaling factor. The result is a sequence of horizontal strokes which can be rapidly displayed. The time to display a full screen-sized letter by this method, doing scaling on the fly, is about half a second at present.

Of course, the result is a hard-edged one bit font, not a smooth soft font. There are many possible methods of producing soft fonts with run-length encoding, but all have several common features. Some sort of filtering must be performed to generate the required grey levels. Unfortunately, filters are usually very slow, so operation on the fly will probably be unacceptable. One possibility is to filter and scale the masters once to produce a soft font data base of the desired size, and then to display them as needed.

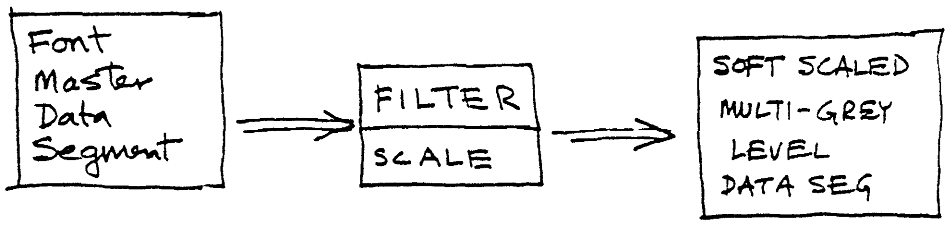


Fig. 1.7

Professor MacNeil has pointed out that for very small type this method will not work due to the fact that one pixel on the eventual screen display covers more than one character and thus the characters cannot be displayed independently.

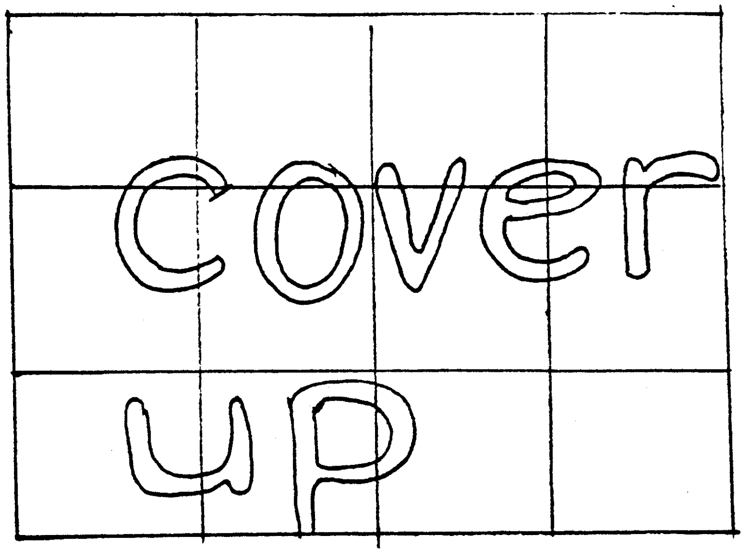
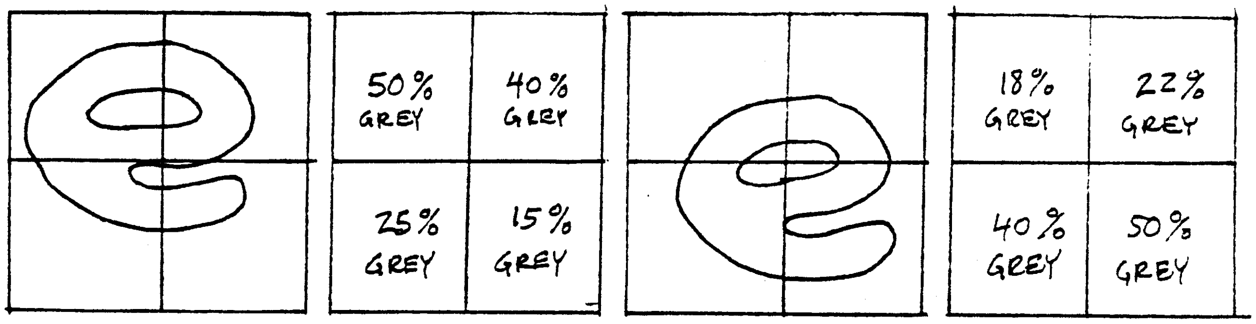


Fig. 1.8

As shown in Fig. 1.8, the letters v and e in the word "cover" both influence common pixels. However, since the blackness of a pixel is proportional to the ratio of black to white area, and for normal type, the characters do not overlap, the proportional cover of the pixel by the letter e can be added to the proportional cover of the pixel by the letter v to yield the true greytone for the pixel. This means that if the drawing algorithm for the precomputed soft character added the new value of grey with the existing value at each pixel instead of blindly overwriting that pixel, the correct effect would be achieved. There is yet another problem, however.

Any character appears differently on the raster depending on its exact position on a <u>sub</u>-pixel scale. For example, the letter 'e' would correspond to different grey level array depending on where precisely the letter was positioned.



Two different grey level arrays for different sub-pixel positions of the letter 'e'.

Fig. 1.9

Fig. 1.9 shows an example of this. Now it is believed by the author that since the total range of error applies only over a single pixel at worst, it is not crucial that this factor in the determination of grey levels be exactly computed. In other words, if for small type, say four different grey scale arrays were generated corresponding to four possible sub-pixel spacings, this would be adequate for the purposes of previsualization. This has not yet been experimentally proven.

A second solution to this problem is proposed. Assuming that four sub-pixel character positions are adequate and that the exact grey level determination need only be computed to two bits of accuracy, the following method will work. (If a three bit (8 level) accuracy is desired then this method can easily be elaborated to achieve it although it will run more slowly.)

Instead of generating soft grey level arrays for each character, (or rather four of them if the sub-pixel spacing ideas are used) and then writing them to the raster buffer as needed to set a line of text, the problem could be looked at differently.

Assume for the present that each pixel must have a grey level accurate to one of five levels, then an array of four sub-pixels can be associated with it. If each of the four sub-pixels are loaded with the correct black or white (one bit) level, then the average of the four sub-pixels can be taken as the grey value of the original pixel.

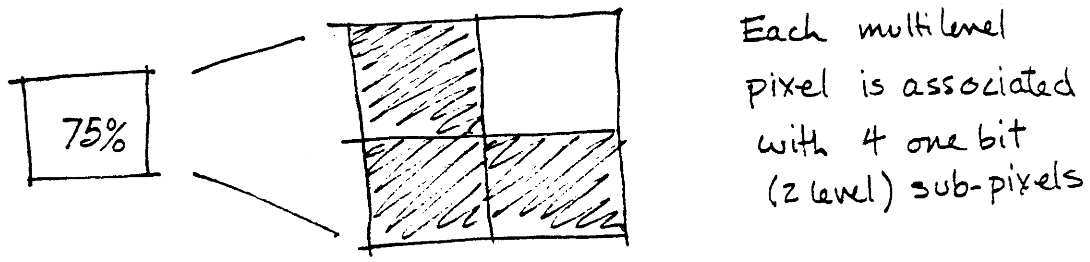

75%
Each multilevel pixel is associated with 4 one bit (2 level) sub-pixels

Fig. 1.10

In this way, the problem of grey level determination is reduced to four black and white decisions. Since all of the characters are resident in core, the display routine can rapidly access the appropriate characters' one bit (black/white) run tables and determine whether or not that sub-pixel is white or black. The four sub-pixel levels are added up (where 1 is black, 0 is white) and divided by four to determine the fractional grey level between zero and one. In practice, it is probably faster to break down a row of pixels into two double length rows of sub-pixels, and then write the resulting row out at once.

CHAPTER TWO

The User Interface

Page layout is a design process. A system to aid the design process must not force the user into fixed methodologies to achieve desired goals. A strict and regid interface between the user and the machine severely impairs creativity. Unless an extremely well thought out structure of control exists, too much time and thought is wasted on deciding which button or key to push next. An interface is presented which offers the advantages of straightforward protocols, self-documentation and personalization. Furthermore, little prior knowledge is assumed on the part of the user.

The interface implementation is based on the application of Pask's conversation theory[5] to interface design as presented by Pangaro.[6]

Consider the topics of a compass, circle, and plane. Clearly the concept of a circle is derivable from the concepts of compass and plane. This is diagrammed as follows:

$$\begin{tikzcd} & \text{circle} & \\ \text{compass} \arrow[ur] & & \text{plane} \arrow[ul] \end{tikzcd}$$

Fig. 2.1

The concept of compass is similarly derivable from the concepts of a circle and the concept of a plane and the concept of a plane is derivable from the concept of a circle and the concept of a compass. Diagramatically this means that the following entailments exist:

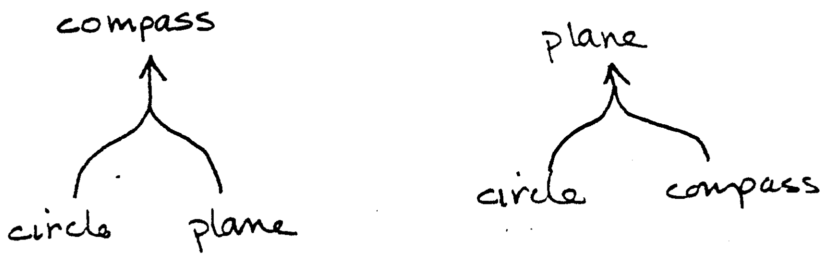


Fig. 2.2

Now if all of these possible entailments exist for a group of topics, the result can be diagrammed as follows:

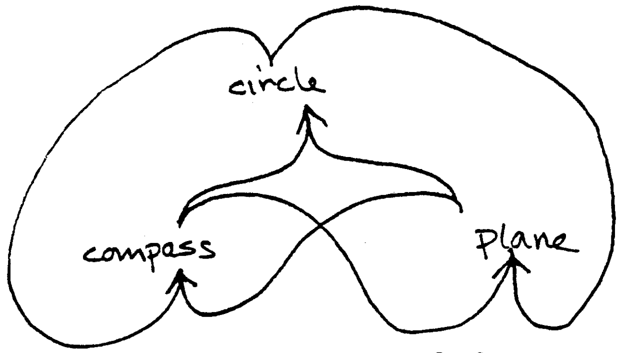


Fig. 2.3

Such a bundle or block is termed "coherent" by Pask. Following these ideas one step further, if topics within one block are derivable from topics in other blocks, then those entailments can also be produced wherein the whole set of topic interconnections is termed an "entailment mesh."

The word "derivable" was used repeatedly in the above discussion. This term may be interpreted in two ways. The first is in a procedural context. Procedurally speaking, one can indeed produce a circle by the application of the procedure of using a compass and the procedure of using a plane (which in this case is held horizontal and stationary). The other context is a conceptual one. One can in fact derive the concept of a circle from the concepts of a compass and the concepts of a plane.

Pangaro's "do-what-do" idea rests on these two definitions of the word derivable. His system allows the user to interact in one of two distinct ways with the topics in an entailment mesh. The user may either "do" a topic or "what" a topic. In either case the system responds by "pruning" the mesh to produce an "ordered derivation" of topics which has as the head node, the topic in question, and which has the topics from which that topic is derived as lower topics.

For example, the pruning of the mesh in Figure 2.4a under the node results in the ordered derivation shown in Figure 2.4b.

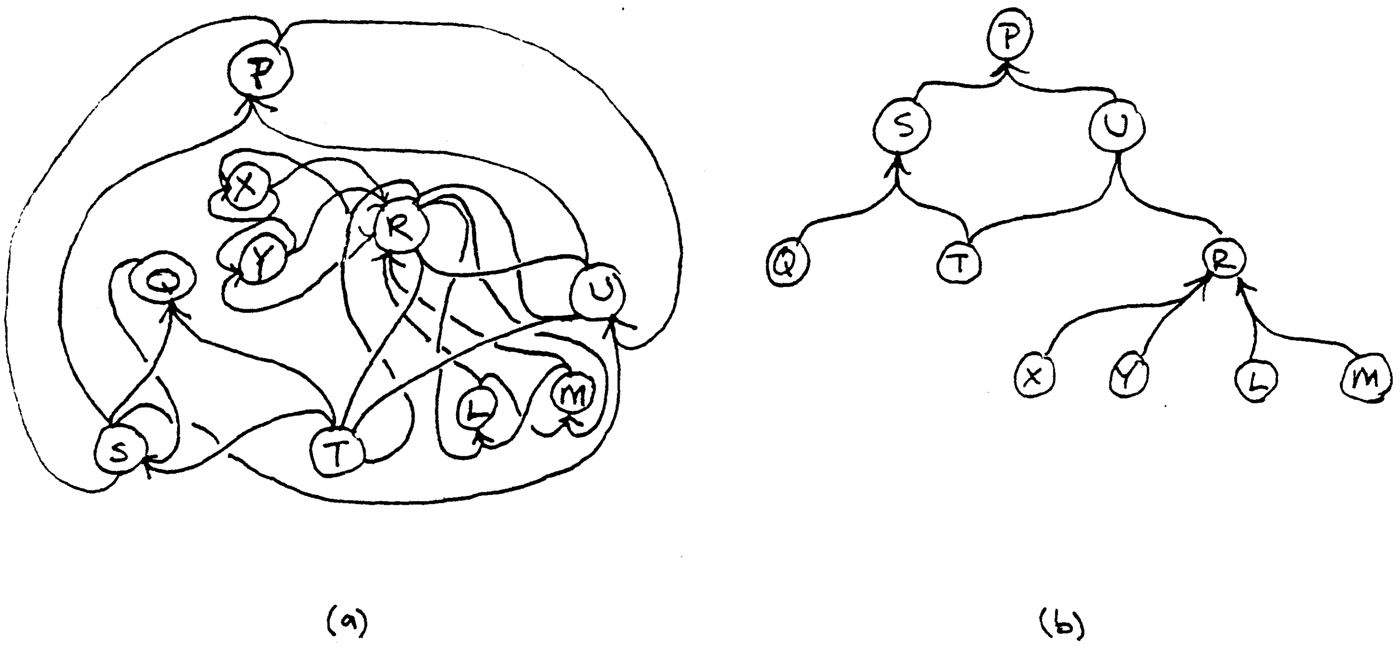


Fig. 2.4

Now, if the "what" context was selected, the system will generate a tutorial about that particular topic from the concepts below it in the ordered derivation, and those concepts will be described in terms of lower concepts, and so on. The infinite regression stops when a topic is encountered which the user already understands.

In the derivation shown in Figure 2.4b, there are two ways of achieving a description of the topic R . The two equivalent derivations can be thought of as alternate methods of achieving the description of R. Whichever method involves the least number of unknown topics is considered the first one to try. If the resultant tutorial is not understood, the alternate is given.

If the context "do" was selected, the ordered derivation would be used differently. The procedural definition of derivable would be used. That is, the topic R would attempt to be achieved by the invocation of the procedures of L and and M, and these would in turn be achieved by invocation of lower topic/procedures. Again, if there are two methods, the most apparently expedient one will be used first.

Needless to say, the implementation of these ideas in existing technology would be difficult. What has been implemented appears to function in the manner described, but in reality uses the crutches of contemporary software techniques to attempt an imitation of the real thing by the use of pre-generated tutorials (written by a human), the use of pre-written procedures, by limiting the mesh to a very few nodes, and by precomputing the prunings.

The applicability of do-what-do to page layout rests on the notion that if a coherent epistomology for the concepts involved in page layout can be constructed as an entailment mesh, the mesh can drive the "what" and "do" functions of the system. The "what" function of the system provides specific tutorials <u>based on what concepts the user already understands</u>. The best system to control a set of procedures is one which is driven by this same mesh, allowing procedures to be invoked to achieve goals in the most straightforward and consistent manner possible. The reason that the operation is guaranteed to be straightforward and consistent is that the mesh which drives the procedural side of do-what-do (the do side, if you will) is the same mesh which is guaranteed by the stringent rules of coherence to describe the process/topic at hand, i.e., page layout. Page layout is in and of itself both a topic embodying a whole wealth of coherent concept blocks, and an equally coherent set of processes.

It is beyond the scope of this paper to either try to prove what is stated above or to delve any further into the theory of entailment meshes, do-what-do, etc. What follows is a description of the author's implementation of a do-what-do-like system as the user interface for a page layout system.

An Implementation Of Do-what-do.

The ordered derivation shown in Figure 2.5 is derived from the mesh used in the layout system. The top node is "signature," which is a group of 16 pages. This is the ultimate goal of the system -- the preparation of 16 pages of text and imagery. Notice this tree is purely hierarchical.

When the user sits down at the system, this tree appears on one of the monitors. By pointing with the tablet at either the "do" or the "what" button, followed by one of the topics, (which are also buttons), he controls the layout system. There are only two types of commands, the "do" command type and the "what" command type.

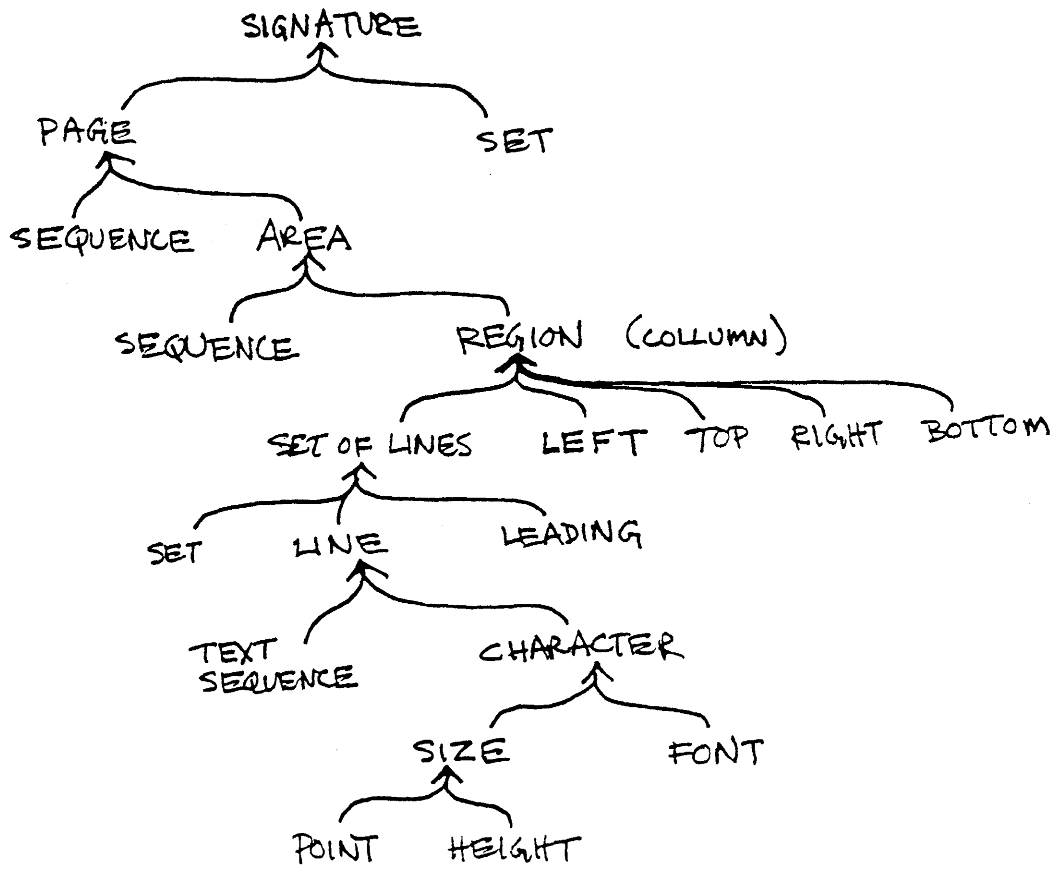


Figure 2.5

The screen appears as shown in figure 2.6. The topic nodes appear to be one of four colors. They have the following meanings:

blue:	inactive, stopped.
pink:	blocked, waiting for some lower goal to be achieved.
red:	ready, running on a virtual processor.
orange:	running, running on a physical processor.

The terms blocked, ready, and running have their standard computer science process control meanings as follows:

Blocked:	means that that topic's process is waiting for some signal. In this case it means that the required lower goals/topics have not all yet been achieved.
Ready:	means that this topic's process would like to run if only there were enough physical processors to go around.
Running:	means that this topic's process is actually running on the one and only physical processor in our system.

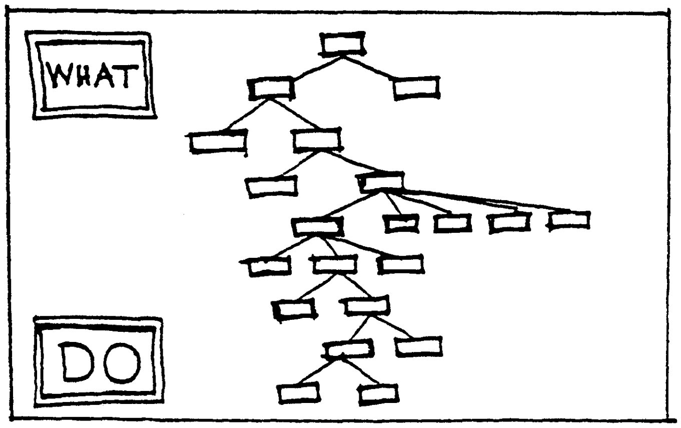


Fig. 2.6

In addition (see figure 2.7), there is a strip of yellow on the bottom of the topic rectangle if that topic has not yet been learned through a "what" command. There is a bar of yellow on the top of each topic rectangle which is currently under automatic control, i.e., this topic function has no meaning when the user pushes the "do" button followed by this topic. The system will run these nodes when all of the required sub-goals have been achieved. The white bars on either side of the rectangle are prompts to indicate that this topic has yet to be achieved.

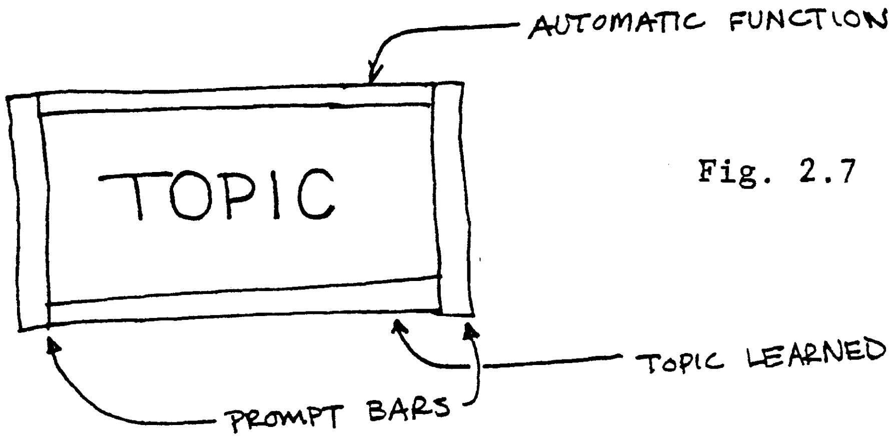


Fig. 2.7

Example of an Interaction

The user sits down at the system and sees all of the nodes in blue (let's say). The user wishes to make a region (collumn) of type. He will hit the "do" button followed by the "region" button. The automatic function bar is indicated above the topic "SET" (of lines), meaning that the computer will determine how many lines are required to set this column. The system responds by placing white prompt bars on the topics below REGION which must be achieved before a region can be made. These prompt bars will stay on only until the user makes another action of any kind. They act as suggestion lights

Perhaps the user does not know what SIZE means. He would then push "what" followed by SIZE. The system responds with a partial pruning of the mesh under the node SIZE. The screen would display this pruning as shown in figure 2.8.

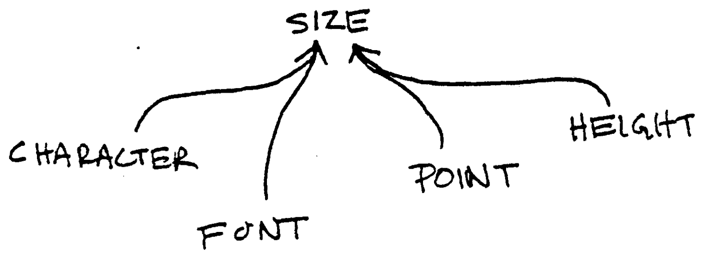


Fig. 2.8

The then selects of the two methods of explaining the concept of SIZE. If the explanation in terms of CHARACTERS and FONT is chosen, a tutorial like that shown in figure 2.9 would appear on the display.

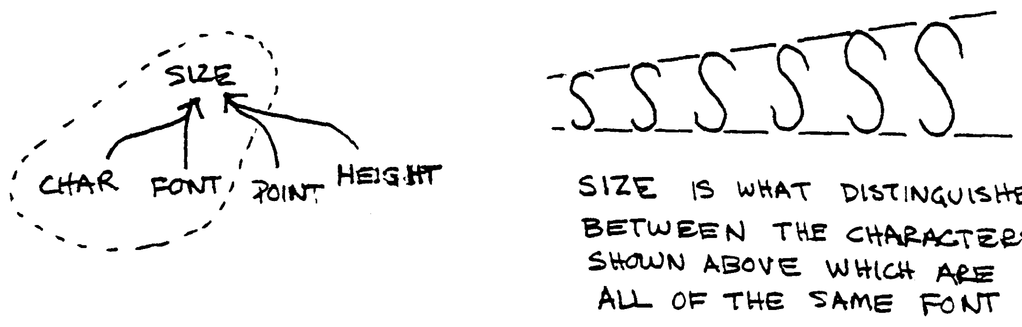

SIZE
CHAR
FONT
POINT
HEIGHT
SIZE IS WHAT DISTINGUISHES BETWEEN THE CHARACTERS SHOWN ABOVE WHICH ARE ALL OF THE SAME FONT

Fig. 2.9

If this is not understood, the user might choose to have it explained in terms of HEIGHT and POINT. In a fully developed system the user would be able to push "what" on any term used in any tutorial and get a tutorial on that topic however in a prototypical system this is not possible.

If the user now presses "do" followed by SIZE, the system turns that topic orange and asks the user to type the desired size on the keyboard. After the new size has been stored by the system, the SIZE topic will return to blue. In a similar manner the font, leading and text input are specified. When each of these topics has been completed then all of the required sub-goals under SET OF LINE will have been achieved. Since SET OF LINE was on automatic, the system will automatically begin the process of setting type into collumn, since all of the required information has been provided. So the net result of the whole process will be a previsualization of a collumn of type on the other monitor.

The Overlord

The code which runs the whole user interface is called the overlord. It watches the user's tablet motions and calls the required function.

The data structure for the mesh is kept in its own data segment. There are two main components are the topic dictionary and the block list. Each topic consists of a PL/1 data structure which includes the name of the topic, a pointer to the next topic, the x,y position, the current state of the topic (blocked,running etc.), whether the topic has been learned, and pointers to parameters which must be passed to the topic's process. There is also a pointer to the topic's own parameter structure. The topics in the dictionary are linked into a list as shown in figure 2.10.

Blocks are stored as a two level linked list. Each block structure points to a list of the "topicblocks" in that block as shown in figure 2.11. The topicblocks form a list of the topics contained in that block. They are analogous to cons

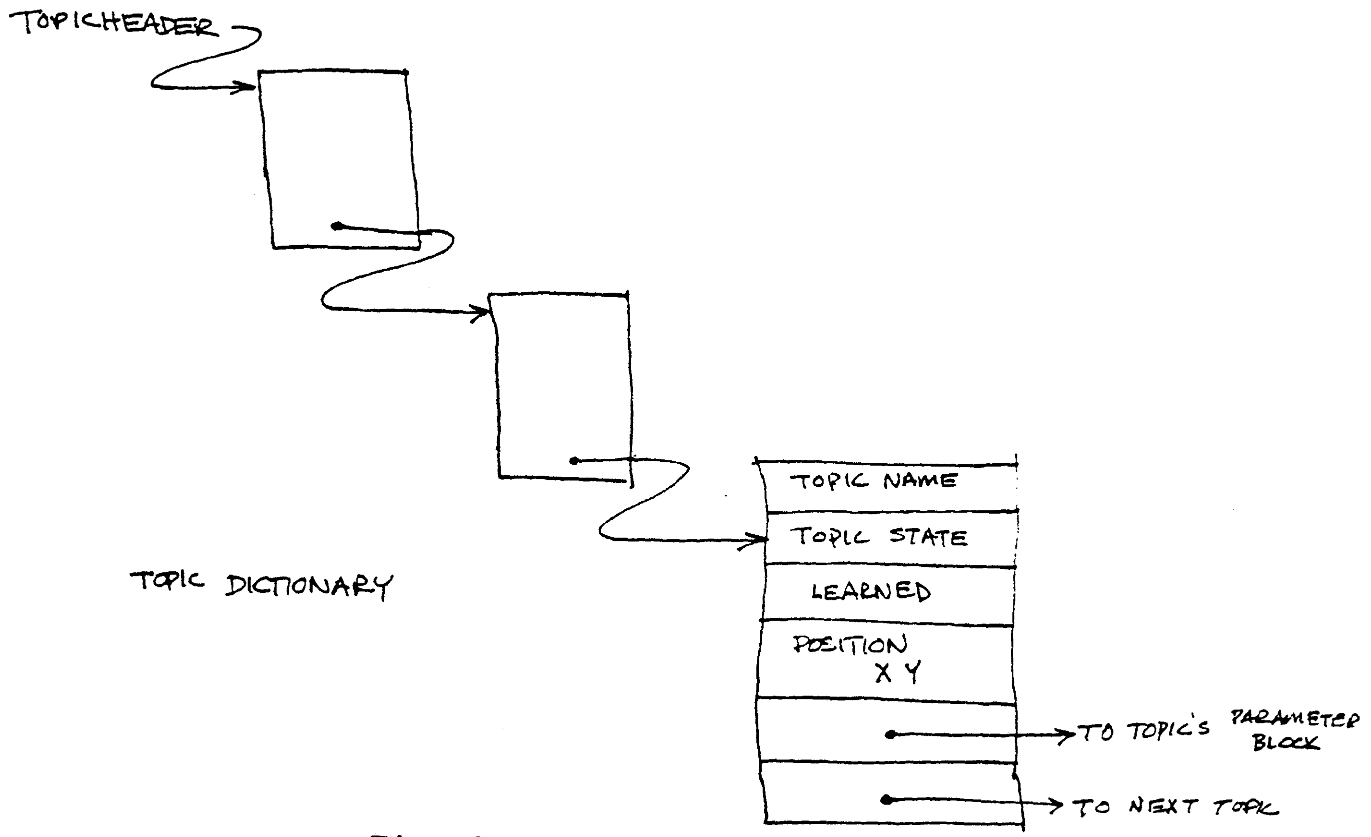


Fig. 2.10

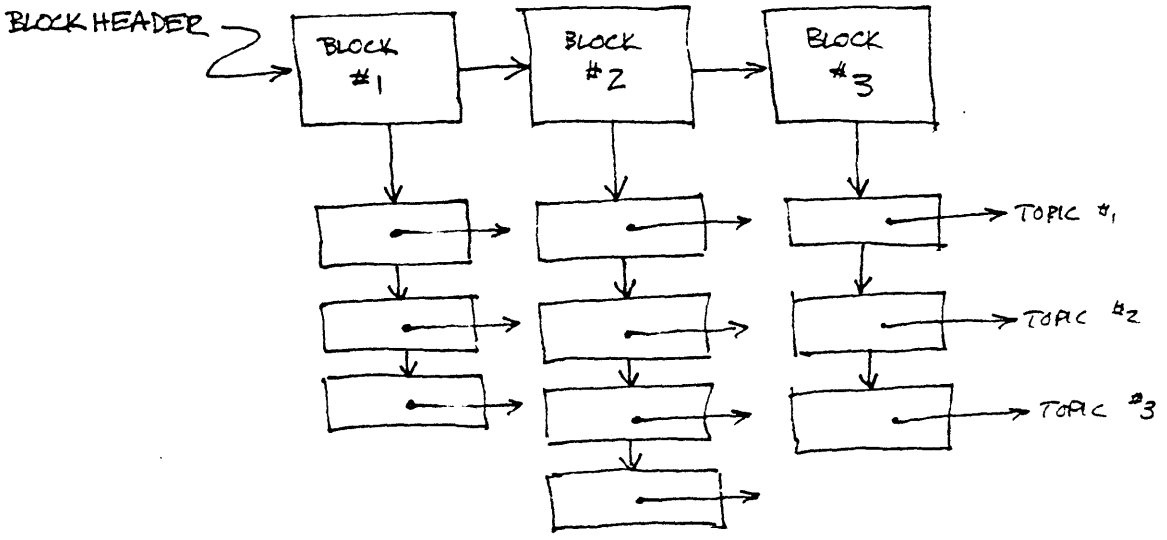


Fig. 2.11

cells in Lisp. Each topicblock contains a pointer into the topic dictionary and a pointer to the next topicblock in the chain.

A set of programs called "Tstick" manipulates these topic and block lists, asks the user for parameter in the topic structure, and builds the mesh data base. These routines are derived from a Pascal implementation of Pask's THOUGHTSTICKER.[7]

Each topic can be thought of as a process. A topic process can be in any one of four states at any one time. Figure 2.12 shows a state transition graph for a topic process.

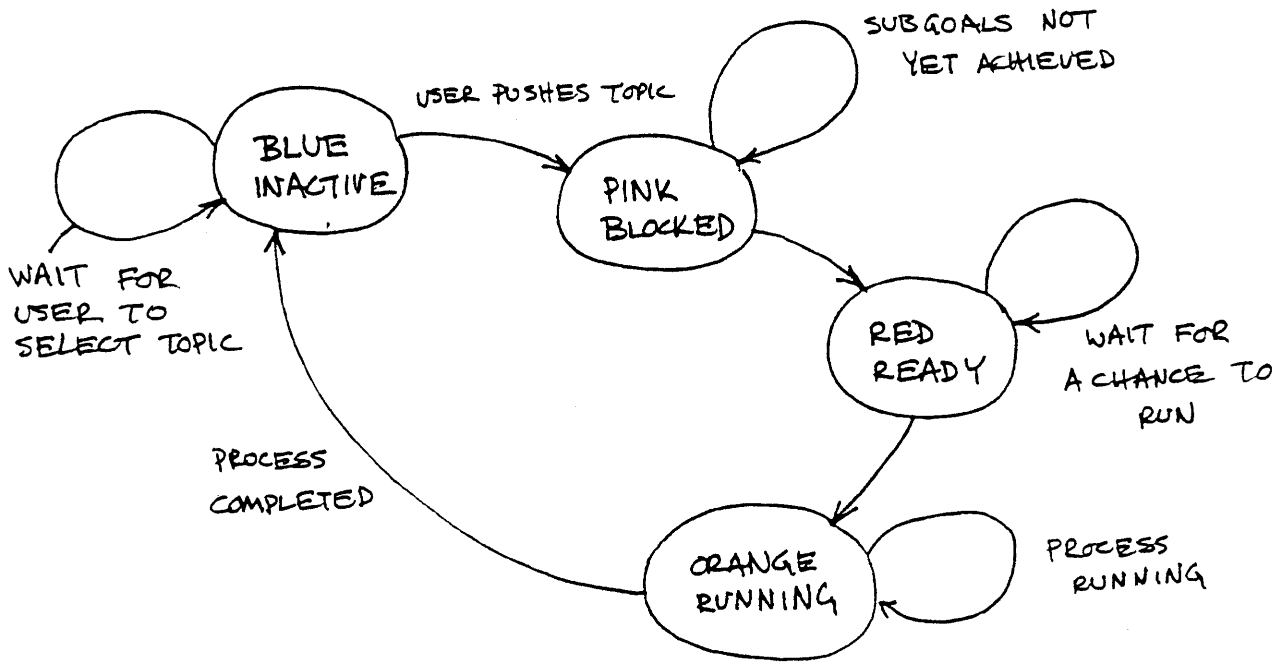


Fig. 2.12

If the topic process is in the blue-inactive state and the user pushes "do" followed by that topic, it moves to the pink-blocked state, and is placed on a list of blocked topics. The user sees the topic he has pushed turn from blue to pink. The list of blocked topics is scanned to see if any of the topics are ready to run. If they are, they are move to the red-ready state and on the orange-running state when actually running.

In the present overlord implementation, independent processes in the computer science sense are not used. Instead a sequential list of subroutines is used. This does not allow simultaneous processes, however at the present there is no need for two topic processes to be running at the same time. To signal when one process has been completed, a bit is set in its topic structure. This is instead of true interprocess signalling, but since there is only really one virtual process, this was not necessary.

To actually run a topic, a call is made to a piece of code by the same name. If one considers each topic and its associated parameter data structure to be the data elements in a new language, then the topic code plays the role of an "actor" in computer science terminology. The topic code is a topic handler and can be thought of as a data handler in a data driven language.

Since it is necessary to create many instances of columns, areas, pages, images, text lines and characters, some mechanism must exist to keep track of exactly which instance is being created or manipulated. The SET and SEQUENCE nodes perform this function. As a side effect of running "DO-SEQUENCE," all of the "topic completed" bits are reset in the topics which are associated with that SEQUENCE node.

For example, if it is desired that a new column be created and added to the sequence of columns already created, the SEQUENCE node associated with the REGION (column) node is pressed. This causes the next column created when "DO-REGION" is pressed to be chained onto a list of already created columns.

The SEQUENCE and SET nodes are also used to allow the user to review what has been created and/or to delete old instances. An elegant way of configuring these nodes as concurrent processes to perform the set building operations has not yet been found.

CONCLUSION

As the technology of computer text formatting and page layout systems improve, increasing emphasis will be placed on the design of the user interface. The first chapter of this thesis showed how immediate full color previsualization of a page of text and imagery is possible. This type of display will make the process of text and image handling less error prone and undoubtedly less frustrating. It will become no longer necessary to look at the final output to make aesthetic and technical judgements on the result which are then used to produce the second attempt. Instead, the raster display will provide an image of the result before it has even been produced allowing the changes to be made interactively.

The interface methodology presented in chapter two provides the advantages of self-documentation, rapid learning rate and ease of use at the expense of some rather advanced, and at present still experimental programming. Much work remains to be done before such a system becomes operational even in the laboratory. At present only the simplest parts of the system have been implemented.

Work continues at the VLW on both of these aspects of text and image handling systems.

REFERENCES

1. Frank Crow, "The Aliasing Problem in Computer Generated Shaded Images," SIGGRAPH 20(11), 1977.

2. Donald Knuth, "TEX and METAFONT: New Directions in Typesetting," American Mathematical Society, 1979.

3. Jonathon Seybold, The Seybold Report, #24, 1979?

4. Christopher Schmandt, "Soft Fonts," IFIP, 1980.

5. Gordon Pask, Conversation Theory: Applications in Education and Epistemology, Elsevier, Amsterdam, 1976.

6. Paul Pangaro, personal communication, 1981.

7. Gordon Pask, "An Essay on the Kinetics of Language, Behaviour, and Thought," SRL, 1979?

BIBLIOGRAPHY

Crow, Frank., "The Aliasing Problem in Computer Generated Shaded Images," SIGGRAPH 20(11), 1977

Goodstein, David., "Output Alternatives," Datamation 26(2), 1980.

Ilson, Richard., "An Integrated Approach to Formatted Document Design," Master's thesis dept. of Electrical Engineering and Computer Science, M.I.T., August 1980.

Knuth, Donald E., "TEX and METAFONT: New Directions in Typesetting," American Mathematical Society and Digital Press, 1979.

Leler, William J., "Human Vision, Anti-aliasing, and the Cheap 4,000 Line Display," SIGGRAPH 14(3), 1980.

Marcus, Aaron, "Computer-assisted Chart Making from the Graphic Designer's Perspective," SIGGRAPH 14(3), 1980.

Negroponte, Nicholas, "Raster Scan Approaches to Computer Graphics," Architecture Machine Group, working paper M.I.T., 1976?

Newman, William M. and Sproull, Robert F., Principles of Interactive Computer Graphics, McGraw Hill, 1973.

Pangaro, Paul, "Thoughtstiker Implementation Note," Systems Research Ltd., Richmond, Surrey, April 1980.

Pask, Gordon., "An Essay on the Kinetics of Language, Behaviour and Thought," Systems Research Ltd., Richmond, Surrey, England, 1979?

Pask, Gordon., Conversation Theory: Applications in Education and Epistemology, Elsevier Publishing Co., Amsterdam and New York, 1976.

Pask, G., "Conversational Techniques In the Study and Practice of Education," British Journal of Educational Psychology, vol. 46 pp. 12-25, 1976.

Pringle, A., Robinson, P., and Wiseman, N., "Aspects of Quality in the Design and Production of Text," SIGGRAPH , 1979.

Schmandt, Christopher., "Soft Typography," IFIP, 1980.

Seybold, Patricia, B., "Wang's 10A,20 and 30 Word Processing," The Seybold Report, 1(1), 1978.

Tilbrook, David., "A Newspaper Pagination System," Masters thesis dept. of Comp. Sci. University of Toronto, 1976.

Unger, Gerard., "Digitising Type," Paper for Pre-Circulation to Participants at the Cambridge Seminar on Digitisation, Cambridge, England 25-27 June 1980.

Warnock, J. E., "The Display of Characters Using Grey Level Sample Arrays," SIGGRAPH, 14(3), 1980.