NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NON-ROMAN FONT GENERATION VIA
INTERACTIVE COMPUTER GRAPHICS

by

James Claude Artero

June 1986

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) Unclassified
Non-Roman Font Generation Via Interactive Computer Graphics

12 PERSONAL AUTHOR(S)
James Claude Artero

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Masters Thesis	FROM _____ TO _____	1986 June 20	97

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
symbol manipulation, symbol coding, font management, computer graphics, raster refresh graphics, font editor, non-Roman font, signal interpretation

19 ABSTRACT (Continue on reverse if necessary and identify by block number)
This study examines the characteristics of computer symbol manipulation systems, including the conventions governing conversion of input symbols to internal code and back to output symbols. Methods to achieve flexibility in the manipulation of large, non-standard, and non-Roman-character symbol sets are discussed, primarily by examination of word processing systems designed to operate on non-Roman fonts. The intent of this discussion is to highlight the desirability of moving toward a computer design approach with incorporates generalized symbol management capabilities. "Leading-edge" computer graphics systems are then evaluated for their potential to host advanced font management systems resulting from this innovative design approach. Finally, the BUILDFONT Font Creation and Editing System, a software utility implemented on the IRIS-2400 series Graphics Workstation, is presented as a tool to assist researchers to develop generalized symbol management applications.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Michael J. Zyda	22b TELEPHONE (Include Area Code) 408 646-2305 22c OFFICE SYMBOL 52Zk

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Non–Roman Font Generation Via Interactive Computer Graphics

by

James Claude Artero
Lieutenant, United States Navy
A. B., University of California, Berkeley, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1986

ABSTRACT

This study examines the characteristics of computer symbol manipulation systems, including the conventions governing conversion of input symbols to internal code and back to output symbols. Methods to achieve flexibility in the manipulation of large, non-standard, and non-Roman-character symbol sets are discussed, primarily by examination of word processing systems designed to operate on non-Roman fonts. The intent of this discussion is to highlight the desirability of moving toward a computer design approach which incorporates generalized symbol management capabilities. "Leading-edge" computer graphics systems are then evaluated for their potential to host advanced font management systems resulting from this innovative design approach. Finally, the BUILDFONT Font Creation and Editing System, a software utility implemented on the IRIS-2400 series Graphics Workstation, is presented as a tool to assist researchers to develop generalized symbol management applications.

TABLE OF CONTENTS

I. INTRODUCTION

It is a common human endeavor to invent symbols which can be used to represent signals. For example, a repeated, low-pitch horn may stand for a warning that there is heavy fog in the harbor. According to Y. R. Chao, a symbol is "anything, linguistic or non-linguistic, which stands for or 'symbolizes,' something else" [Ref. 1: p. 194]. But to be useful, a symbol must be something which can be conveniently produced, presented, and perceived without necessarily perceiving the object that it stands for. Magnetizations on a recording tape stand for variations in sound, yet they cannot be produced and perceived conveniently without electronic equipment. For human-to-human communication concerning variations in sound, a written music score is a better (i.e., useful) collection of symbols when it is not necessarily desired to produce the sounds themselves.

In popular usage the words "symbol," "sign," and "signal" are used almost interchangeably. Although mathematicians, logicians and psychologists do not agree on the precise meanings of these words, there is differentiation in their meanings when the words are used on a technical level. A puff of smoke coming from a piece of equipment is a *sign* that a malfunction may be imminent. It may also be a *signal* to take immediate corrective action. It is not necessarily a *symbol* for the malfunction. On the other hand, a red light on an instrument panel may be the specific *symbol* for an imminent malfunction. In this case and in the case

of the fog horn mentioned above, the *symbol* is also a *signal*. By some, usually arbitrary, convention which established the relationship beforehand, a *symbol* is used to represent some object or concept. An instance of actually using the *symbol* may then become a *signal*. By a convention of the English written language, the word "halt!" is a symbol for the concept of demanding an abrupt stop. When "halt!" is spoken in a specific situation, it becomes a signal to make the abrupt stop. [Ref. 1: pp. 194-195]

Inventing and employing symbols that can be used to represent signals is a practice which preceded the development of computer science by several thousand years. Nevertheless, computer science relies upon this practice extensively. Data is presented to a computer in the form of input symbols. The computer user passes a string of symbols into the computer, and at some point these symbols are converted into signals which the processor can interpret. The logical point where conversion from symbol to signal takes place is the interface between the user and the processor, and it must include the convention which defines how the symbols will be coded into signals. Another, similar convention must have been established for the output side, so that when the processing is complete, the processed information can be reformulated into symbols meaningful in the external (user's) environment. All computer installations have this symbol-to-signal conversion capability associated with their input and output devices.

Once the input stream is properly coded into signals for the internal use of the processing elements, we want to be able to assign interpretations to these

signals. Depending on how the signals are combined and recombined, many interpretations are possible. The computer can be made to perform many different tasks according to these signal interpretations. It is an objective of computer science to maximize the number of possible interpretations--to maximize the variety of tasks which can be performed by a computer, and to make the computer become a truly general-purpose machine.

The purpose of this study is to investigate and support methods to generalize conventions controlling conversion between symbol and internal code. One approach is to set the number of discrete codes to some arbitrary size and then work on abstracting the principles governing the way combinations of these codes can be interpreted. Following this approach, computer scientists from the United States and Europe have developed two standard sets of symbols to facilitate the encoding of user input. The ASCII (American Standard Code for Information Interchange) set includes 128 alphanumeric characters, special characters, and non-printable control codes. The EBCDIC (Extended Binary Coded Decimal Interchange Code) set can code up to 256 characters and control signals although not all possible codes are actually used. It can be seen that members of either the ASCII or EBCDIC symbol sets are easily coded in a single eight-bit byte.

A second approach is to remove the size limitation from the set of input symbols, allowing the user to choose any number and form of symbols from which to compose his input. With this latter approach, the objective of improving and generalizing computing capability is realized by utilizing many more codes, not

(as with the previous approach) assigning interpretations to many combinations of a limited number of codes. However, having more symbols to choose from also leads to many possible interpretations, and so the two approaches are in fact different means to the same end.

The first approach, that of using a "standard set" of symbols, is well established. Agreement on standard character coding conventions has made it possible for different manufacturers' equipment to interface and operate together compatibly. Clearly, if there were no established standards, every computer facility would use a different symbol coding convention, forcing users to "re-invent the wheel" whenever they moved into a different computing environment. Thus, if not pursued carefully, the greater generality promised by the second approach may be achieved only at the expense of eliminating existing system compatibility. This may be too great a price to pay.

This study opens with a discussion of computer application areas where general symbol manipulation research can produce beneficial results without necessarily destroying established symbol coding standards. The study goes on to introduce and describe the "BUILDFONT" Font Creation and Editing System, a software utility developed for the IRIS-2400 Graphics Workstation. The BUILDFONT System is a tool intended to assist researchers who will address the task of improving symbol handling generality.

II. BEYOND STANDARD CHARACTER SETS

A. APPLICATION AREAS FOR NON-STANDARD SYMBOL SETS

Although the ASCII and EBCDIC sets are perhaps the most familiar (at least in Western culture), there are other symbol sets which are also recognized as standards. For example, the "Code of the Japanese Graphic Character Set for Information Interchange" (JIS C 6226 1978) is the Japanese standard for input and output of Japanese (*kana* and *kanji*) characters. The Chinese have three standards: "Information Exchange for Chinese Character Codes (Basic Volume)" (GB 2312-80) (used in the PRC), the Chinese Cable Department's "Standard Cable Code" (used in the PRC), and "Chinese Character Code for Information Interchange" (used in the ROC). Obviously, progress in the computer hardware and software manufacturing industries would have been much (painfully) slower without acceptance of these standards. However, agreement on standards does not solve many problems relating to generalization of computing systems. General computing hardware can be more flexible and can find wider application if we can abstract symbol set manipulation techniques to address the following objectives:

- ☞ Customizing character fonts for a given symbol set to provide a variety of user-specified forms, or type sets, for any particular symbol in the set.

- ☞ Accommodating rapid definition and design of symbol sets containing an arbitrary--perhaps very large--number of symbols.

A user-friendly font editor, such as the BUILDFONT System described in this study, satisfies the first objective above. Success in achieving a general methodology for the second objective promises to facilitate some interesting possibilities for new computer applications, among them:

- Providing a way to adapt existing computer installations to handle customized symbol sets used in a newly developed programming language (rather than having to adapt the language to the existing symbol set). For example, functional programming languages may well become regarded as the next generation of computer languages since they offer hope for developing methodologies for programming highly parallel (5th generation) computer architectures. Thus, developing a better capability to process these languages may be necessary for the productive exploitation of VLSI technology.

- Bridging the symbology barrier between American word processing technology and the word processing needs of countries which do not use Roman alphanumeric symbols to write their languages (e.g., Chinese, Japanese, Korean, Arabic, Cyrillic, Greek, etc.).

- With computer supported models, a system of graphics objects are often employed to provide the user with a visual analogy of the modeled world. If these objects can be represented consistently by elements of a symbol set, then the set may be stored and retrieved as a customized character font, simplifying the programming task. The Navy Tactical Data System (NTDS) is an example of this application area.

- In the field of cryptology, the ability to rapidly define and redesignate symbol sets of arbitrary size and complexity has many applications.

B. BACKGROUND REVIEW OF THE WORD PROCESSING APPLICATION

Of the possible application areas described above, word processing with non-standard symbol sets is our main focus. The reason for this choice is that the word processing application best represents the dimensions of the problems we are

trying to solve by developing generalized techniques for symbol manipulation. Specifically, word processing with non-standard symbol sets demands the capability to customize character fonts. Additionally, the various symbol sets which have evolved to facilitate written language are arbitrary in size--indeed, some are very large (see Chapter 3).

English and the oriental languages, Chinese, Japanese and Korean, express their respective written languages by means of symbol sets which differ greatly in the number and form of the symbols they contain. How can computing systems designed for the support of one character set, be generalized to the extent needed to support any other set? To answer this question, we need to consider the way that each of the languages mentioned uses symbols to capture its meaning in written form.

Let us compare and contrast some linguistic properties of English, Chinese and Japanese, which affect machine processing of these languages. Here we need to discuss a hypothetical "standard" dialect of each language. For this, we select the dialect taught in schools, prescribed by the national or regional government, etc. We avoid totally issues of whether anyone actually speaks the standard dialect.

1. English

Of the natural languages we discuss, English is the richest from a phonetic standpoint. The Merriam-Webster unabridged dictionary lists 22 distinct symbols needed to transcribe vowels and another 28 for consonants. This

results in a staggering number of combinatorial possibilities for different sounding syllables. Actually, most linguistics experts agree that there are at least ten vowels (including diphthong vowels) in standard English [Ref. 1: pp. 27-31], and over 10,000 distinct phonetic combinations (syllables).[1] Since written English is based on phonetic transcription, the 26 alphabetic characters must be able to express all of these syllables. Phonetic transcription of English by means of the standard alphabetic characters is very imprecise and inconsistent. For example, the ten-plus vowels must be transcribed by using only six symbols: a, e, i, o, u, y. Fortunately, the education system is able to drum the exception handling rules into students, and thus native speakers of English get along fine with the 26 alphabetic symbols (although some of them turn out to be notoriously bad spellers). The compactness of the character set and the ability of native English speakers to deal with inconsistencies in its use, i.e., the establishment of phonetic transcription conventions in English, has fostered a long and successful development of English language word processing. The emergence of the "Qwerty" typewriter keyboard in the early 1900's established a format for the standard word processing input device. With the addition of special purpose keys, the Qwerty arrangement has been adopted as an input device for computing equipment, and thus the transition to electronic word processing has been easily achieved for a generation of English-speaking touch typists.

[1] Although English is a "word-unit" language, the analysis here discusses numbers of syllables used in order to facilitate comparison with Chinese and Japanese.

In summary, the essential characteristic of English language word processing is the capability to handle a complex natural language phonetic system with a relatively small symbol set. The computer, of course, knows nothing about the complexity of the language it is processing; it knows only that the symbol set is (or is not) within its storage and manipulation capacity.

2. Chinese

The situation with Chinese word processing differs greatly from that of English. Curiously, there is some similarity in the linguistics of the two spoken languages. On a structural or syntactic level, Chinese and English are so-called "analytic" languages, relying more upon position of a word (in the sentence) than upon specialized prefixes or suffixes to determine grammatical category [Ref. 1: p. 69]. On the phonetic level, both languages use about the same number of vowels. However, the phonetic rules of Chinese are much more structured and regular, and there are fewer consonants available. There is a greater discipline imposed on the formation of syllables in Chinese phonetics: They are formed of "initials" (a set of optional consonants), "medials" (a very restricted set of optional vowels), and "finals" (a required combination of vowels or vowels and ending consonants). In addition, each syllable has a "tone" associated with it. The tone, a prescribed variation in pitch used when the syllable is pronounced, is a phonetic element that further differentiates the possible meaning of the syllable. There are four possible tones plus "neutral" (no inflection) in Mandarin Chinese.

Due to the restrictions and regularity of Chinese phonetics, there are only about 1300 discrete syllables in the language [Ref. 1: p. 210 and Ref. 2: p. 20] compared to the 10,000-plus of English. It would appear that Chinese would lend itself to phonetic transcription much more easily than English. Indeed, this is the case. Standard Mandarin Chinese has an excellent set of phonetic transcription symbols, the National Phonetic System (in Chinese, *Zhuyin-fuhao, Zhuyin-zimu*, or (slang) *Bopomofo*). In this system, each initial, medial and final has a unique symbol. A set of thirty-seven characters is sufficient to represent, with three or fewer phonetic symbols, any syllable occurring in the spoken language. This can be accomplished without the ambiguity or inconsistency of written English. [cf. Ref. 2: p.18, Ref. 3: pp.32-33, and Ref. 4: pp. 141-145]

The symbols of the Chinese National Phonetic System can be represented by the Roman alphabet as well. There are a number of systems for doing this, however, the government of the Peoples' Republic of China (PRC) has sanctioned only one: the *Hanyu-pinyin*[2] System [Ref. 3: p. 32]. Thus, Chinese can be phonetically transcribed quite easily and accurately utilizing the same well-established word processing devices developed for English.[3]

Unfortunately, whereas (somewhat imperfect) phonetic transcription methods solve the entire word processing problem for English, these same

[2]The glossary (Appendix A) contains names of some of the other popular or historically important systems.

[3]Actually, although a Qwerty keyboard may be adapted for this purpose, the character use frequency of Romanized Chinese is quite different from English and European languages, and this fact justifies keyboard redesign, as we discuss in the following section.

methods only scratch the surface of the input problem with Chinese word processing. Written Chinese is not a phonetic record of the spoken Chinese language, and there is very little relationship between phonetics and the construction of Chinese textual material. There are about 50,000 (largely non-phonetic) symbols (called *hanzi* [4]) used to express the syllables of written Chinese (although perhaps only about 3000 occur with frequency). Processed written Chinese must be expressed in combinations of these symbols rather than the more workable phonetic character sets. Also, with only 1300 possible syllable pronunciations, it is clear that many characters must have the same pronunciation. Deciding which written character corresponds to a particular instance of a pronunciation is known as the "homophone resolution problem."

Although Chinese civilization is acknowledged as the first to develop mechanical printing techniques, after more than one thousand years of experience with mechanical symbol manipulation, the Chinese have yet to develop a reasonable keyboard-like device equivalent in utility to the Qwerty typewriter. Furthermore, the Chinese are not taught to think in terms of phonetic symbols when they compose written Chinese, and they cannot easily read meaning into streams of phonetic characters which have been printed out by another person or a device. For these reasons, Chinese word processing represents an extreme challenge to the generalization of word processing techniques.

[4] *hanzi* is the Chinese word meaning "Chinese character." It is the same word which is pronounced *kanji* in sino-Japanese. In this study, the form used depends on which form appeared in the source material being discussed.

3. Japanese

Contrasted with analytic natural languages, including English and Chinese, Japanese is a so-called "agglutinative" language [Ref. 1: pp. 87-89 and Ref. 5: p. 38]. Suffixes and verb declinations, rather than word order in a sentence, are very useful in Japanese for determining the grammatical category of a word (there are also a few relatively unimportant prefixes). Thus, despite the fact that the Japanese phonetic system is much more restricted and regular than even Chinese (i.e., the homophone resolution problem exists in Japanese as well), the language can be easily transcribed into streams of phonetic characters which are consistent, unambiguous, and which do not require further embellishment for understanding. The Japanese have excellent *kana* phonetic syllabaries designed to do this. Unfortunately, the Japanese choose to embellish their written language anyway, and this complicates the problem of transcription. The Japanese have continued to use Chinese characters (*kanji*) interspersed with *kana* as a matter of historical development. This practice is analogous to the English use of spellings like "through," which suggest that what is actually being transcribed is an archaic pronunciation. In fact, the written transcription of all languages evolves less rapidly than the corresponding spoken form, and written language often sounds formal and outdated when converted to spoken form (as when a document is read aloud) [Ref. 1: pp. 110-111].

The Japanese spoken language employs five vowels and about 20 consonants to form only about 100 possible syllables. A syllabary of 53 distinct

symbols (*kana*) serves to transcribe the language into written form. Actually, there are two different sets of *kana* representing the same set of sounds: *hiragana*, a cursive script for transcribing native Japanese words (including those borrowed from China); and *katakana*, an angular script for transcribing emphasized words and words borrowed from foreign countries (not including China). The combined 106 symbols of the two *kana* sets form a reasonable alphabet for word processing, but unfortunately written Japanese does not stop there. As we have noted above, Japan has borrowed extensively from the culture of China, and Chinese characters appear normally in the common written language of Japan. As a matter of practice, all words with semantic content (nouns, verb roots, adjectives and adverbs) are expressed in Chinese characters. Therefore, with the exception of input (which can be done using *kana* alone), Japanese word processing must perform character manipulations as challenging in scope as those required by Chinese. The only consolation is that the education ministry of the Japanese government has taken a lead in reducing the number of *kanji* normally appearing in print to a set of 1850 "essential" and "general use" characters [cf. Ref. 6].

C. SUMMARY OF WORD PROCESSING TECHNIQUES FOR CHINESE AND JAPANESE

This section focuses on present word processing technology for written Chinese and Japanese. The discussion applies to the Korean written language as well, but approaches to dealing specifically with Korean were explored by Kim

and Ko [Ref. 7] and Lee [Ref. 8]. The reader who is interested in characteristics of the Korean language as they relate to word processing is referred to those works.

Of techniques which are in use today, most are concerned with the input problem, i.e., how the word processing operator can get the written language properly codified into an unambiguous internal representation within the computer. For example, where input by means of purely phonetic characters is used to designate a stream of actual text (which may or may not be composed of purely phonetic characters), massive software support is needed to replace the input symbols with the final textual symbols that they represent. This is necessary because the phonetic input technique depends on a limited set of input characters (no more than about 50) used in combination to represent any of perhaps 10,000 output characters. Since techniques for output are not specifically incorporated into these systems, they should be called, more properly, "strictly input systems." The systems can be grouped into the following categories:

- ☞ Those based on whole characters (two-dimensional array)
- ☞ Those based on some kind of non-phonetic coding for character input. e.g., TCCM. Yamada two-stroke, "radical"
- ☞ Those based on phonetic transcription
- ☞ Non-mechanical input techniques, e.g., OCR. speech recognition, on-line handwritten character recognition, etc. (Note--these techniques will not be discussed)

A description of the characteristics of some of the more important input systems follows.

1. Input Techniques Based on Whole Characters

a. Touch-sensitive Kanji Typing Tablet

This technique involves a special keyboard, or tablet, which is a descendant of the Chinese Character typewriter (the "Wabun" typewriter invented in Japan in 1913 [Ref. 3: p. 29 and Ref. 9: pp. 37-38]). The tablet is similar to the typesetting rack used for a mechanical printing press. However, instead of selecting a character element from the rack for use by a printing machine, the user merely points to the selected character on the input tablet. A light pen, mouse, or physical touch are ways of doing this. A typical input tablet contains 2000 to 3000 characters, and characters not contained on the tablet must be input by means of a supplementary coding method (one of the other methods described in this section). Since each character desired must be sighted by the operator, this technique is definitely not one which lends itself to touch typing, and input speeds are lower than for other techniques. Character hunting time is proportional to the square root of the number of characters covered. Thus, hunting time is decreased by having fewer characters on the tablet. But input time is increased when a character cannot be found on the tablet at all (and must then be coded for input by other means). Matsuda has determined that 2300 is the optimal number of characters to arrange on the tablet in order to achieve a balance of the factors affecting input speed. [Ref. 5: p. 40]

The principal advantage to this technique is that no extensive operator training is required, other than familiarization with the indexing system

for the character set arranged on the tablet. The cleverness of the arrangement is the only enhancement to input speed available. Any operator can input symbols with this technique, and so it provides a means to do document preparation for the average (untrained) person. Also, no sophisticated or expensive software program is needed to translate input symbols into an internal representation, since there is a one-to-one correspondence between the input symbol and the internal code. The disadvantage of this technique is that it is extremely slow, and the device is highly specific--hardly a generalized input device adapted to a specific purpose.

The touch-sensitive, two-dimensional character array is used in various forms with Chinese, Japanese and Korean. In fact, it is possible to use this input method with any symbol set.

2. Input Methods Involving Non-phonetic Coding

Input via non-phonetic coding seems to be most popular in Chinese word processing. Some experts feel that an arbitrary numerical coding method is best due to the major dialectal differences which exist in China, giving rise to disagreement about which is the best phonetic transcription system. The lack of a clear-cut relationship between the written language and the spoken language has been previously discussed. Ironically, of all the arbitrary coding systems, the one recognized to be most effective, Yamada's "two-stroke method" [cf. Ref. 9], was developed in Japan.

a. The Yamada Two-stroke Input Method

This system uses a limited number of keystrokes to uniquely identify a large number of input symbols. The code for each input symbol is passed into the equipment by making two keystrokes on a special keyboard. Let us say that the input keyboard is equipped with 48 keys (approximately equal to the number of keys, alphanumeric and special, of a standard Qwerty typewriter). Then a two-keystroke input can refer unambiguously to any position of a 48x48 matrix, a total of 2304 possible characters (which is sufficient for about 99 percent of Japanese word processing requirements). Selecting the correct two-keystroke sequence is a basic problem to be overcome in using this technique. The characters are grouped onto each key, so that one keystroke selects the group and another keystroke selects the character within the group. It seems that the amount of memorization required of operators would be an insurmountable obstacle to developing a national pool of competent typists. However, Yamada's research indicates that this is not the case at all. In fact, the method does require intensive training and practice, but the methodology has achieved results comparable to those of English language touch typing. This is because the skill developed is true "finger learning," rather than a conscious level of strenuous mental involvement by the operator. In this respect, the two-stroke method differs from the other arbitrary numerical coding methods (i.e., touch typing is possible).

The two-stroke method, which can theoretically be adapted to any of the oriental languages, represents a training-intensive solution to the word

processing problem for full-time. semi-professional clerical office workers. This solution. unfortunately. is not satisfactory for the occasional word-processing operator who lacks special training and therefore must use one of the other methods.

b. The Three-Corner Coding Method (TCCM)

This technique, developed by a group of computer science professors in Taiwan [cf. Ref. 2], involves assigning three sets of two digits each to every possible *hanzi* of the input set. In other words, each input symbol is represented by a six-digit code. One advantage of the system is that it can be used on a standard ASCII keyboard or even a simple numeric keypad. The method reserves an ample number of code entries for non-Chinese characters, so that the keypad is sufficient for all input. Coding of whole Chinese characters is based on individual two-digit codes for 300 "fundamental symbols" which the system identifies as a component set from which any (complete) *hanzi* can be built. Since the characters are determined uniquely upon input, software support required for internal processing is minimal. Simple table look-up operations suffice. The main disadvantage of the system is that a new, fairly complex, and artificial system must be learned by the operator in order to transcribe the written language. Thus, training time is fairly extensive, and TCCM is not really useful to the non-professional word processor operator.

3. Phonetic Transcription Input Techniques

a. The Morita System for Japanese Word Processing

This system was developed by Masasuke Morita at the NEC Corporation of Japan [cf. Ref. 10]. Although the dominant idea of the system is phonetic transcription of the spoken language combined with software processing and translation into the desired internal representation, the system is somewhat eclectic in that Morita incorporates desirable elements from systems based on other concepts. This is accomplished because of Morita's consideration of the linguistic principles of Japanese and concern for human engineering aspects of the physical input device in addition to focusing attention on the computer science issues relating to word processing.

For the input symbol set, the Morita system uses neither the traditional *kana* syllabaries nor the Roman alphabet as it is laid out (one letter per key) on the standard Qwerty typewriter. Instead, Morita has performed a simple linguistic division based on Japanese phonology so that all pure *kana* input can be accomplished by a two-keystroke sequence of consonants and vowels--one each, to yield the equivalent of a *kana* sound. All *kanji* input can be accomplished by a two-keystroke sequence of initials and finals (initials to include the limited set of medials permitted in Japanese pronunciation of sounds descended from borrowed Chinese syllables). These four types of input elements are arranged as follows: consonants and initials arranged in three rows of five keys each (under the operator's right hand); vowels and finals arranged in three

rows of five keys each (under the operator's left hand). A shift key is provided under each thumb. and all the possible syllables allowed by Japanese phonology can be assembled with no more than two keystrokes (possibility of one opposite thumb shift with each keystroke). Major advantages of this system are that the keyboard is about equivalent in number of keys and arrangement to a standard western typewriter. With three rows of five keys under each hand a "home row" concept is employed which facilitates touch typing. In fact, input speeds can be expected to equal or exceed those obtained by experienced operators of English language word processing equipment. Also. selection of a "finals" key by the left hand not only facilitates a direct phonetic transcription of the spoken Japanese word, but also indicates that the word being assembled is a *kanji* in the written representation. This feature completely eliminates the need for software segmentation[5] and grammatical analysis of the input stream, so that the software support can be concentrated exclusively in the area of homophone resolution. As a result. with a reasonably fast *kanji* look-up and homophone resolution method, input text can appear in its correct, internally represented form instantaneously on the screen. without the necessity of being run through an intermediate translation program.

The two-keystroke-per-syllable input method of the Morita system seems similar to pure arbitrary coding schemes such as the Yamada two-stroke method for Japanese and TCCM for Chinese. Morita's advance over these

[5]See the section on software support below.

systems is that the "arbitrary" coding scheme is not numeric: the phonetic system of the Japanese language is the scheme. The phonetic elements pictured on the keys are expressed as combinations of Roman letters (or *romaji* as the Japanese call them). This is a perfectly correct and logical choice because *romaji* is a system which can faithfully transcribe all Japanese sounds, and it is taught to all Japanese as a standard part of the education system, so users are not required to learn a new coding convention.

In summary, the Morita system allows rapid input of the Japanese language by conventions which are familiar to native speakers, while it requires a minimum of software support to perfect the accuracy of internal representation of the text. Because of these features, minimal operator training is required to develop acceptable proficiency.

On a conceptual level, the Morita method completely solves the input problem for Japanese text. The question which remains is, can general computing equipment support the method efficiently enough to make it workable?

b. Chinese Phonetic Input Keyboards

Two purely phonetic input systems for Chinese word processing are described by Sheng [cf. Ref. 11]. Both systems depend on a set of symbols constructed according to the *Hanyu-pinyin* phonetic system for transcribing standard Chinese. The *pinyin* system was introduced by the PRC about 25 years ago, and it is a learning requirement for students educated under the Chinese educational system. In *pinyin*, initials and finals are expressed in Roman letters.

As Morita has done with his *romaji* keyboard, Sheng's systems place onto each key all of the Roman letters needed to express a single initial or final. Thus, these systems work just as well for any consistent phonetic transcription system (of standard Chinese), including the National Phonetic System which was described in a previous section and which is still the phonetic transcription system used in Taiwan (ROC).

The systems described by Sheng differ only in the layout of input symbols on a standard keyboard configuration (actually Sheng recommends including a "software switch" with the systems, facilitating instant conversion to a western language word processor, if desired). One of these keyboards uses four rows of keys (total of 44 character keys), with shifts required for 31 keys. A total of 58 (complex) phonetic characters and 37 special characters (punctuation, numeric, control, etc.) can be accessed. This keyboard is used with the HZ-80 Word Processing System, introduced at an international computer conference in Hong Kong in 1980.

The other keyboard also uses a standard keyboard configuration, but distributes the complex phonetic characters onto three rows (using a total of 32 keys for these three rows). The fourth row of 12 keys is used for ASCII-type numerals and special characters. In this configuration, the same 58 phonetic characters and 37 other characters can be accessed, but up to 39 shifts must be made. By arranging the phonetic characters into only three rows and placing most special and numeric characters on the fourth (highest) row, phonetic input

of written Chinese is reduced to the proportions of the English language input problem (i.e., "home row" concept and touch typing are facilitated). In addition, this alternative seeks to minimize the effects of local dialectic variations by clustering certain initial and final syllable component characters onto the same key. In this way, when a particular speaker fails to distinguish a phonetic difference present in the "standard" language, the mistake usually can be corrected by shifting and re-striking the same key.

Since the primary input method of these systems is phonetic, the problem of homophone resolution must be dealt with. Some techniques used to accomplish this are discussed in the next section.

4. Oriental Language Word Processing Software Support

Matsuda summarizes the problems associated with word processing support programs:

> In implementing a translation dictionary, the primary issue is how small and cheap to make it. In general, a kana-to-Kanji conversion system using Kanji-designated segmentation requires a dictionary of about 50,000 words (that is, a 5M-bit memory is needed to store it). [Ref. 5: p. 41]

Since we want to develop general methods to facilitate word processing with arbitrarily constructed symbol sets, and we want to be able to apply these methods to any reasonable hardware configuration, including the present generation of desk-top microcomputers, a requirement for 5M-bit memory may be unacceptable. In Chapter 3 we investigate physical memory requirements to

support a generalized system, and to some extent these requirements tend to pull our attempts at abstraction back toward unpleasant realities.

a. Xerox Star and Fuji Xerox J-Star

These systems perform purely phonetic transcription at the "word-unit" level. This concept has been adapted for use with Chinese, Japanese, Korean, and other languages. These systems are described in this section because the role of input translation is the significant characteristic of the systems. A standard-size keyboard can be used for input, and the input alphabet can be either Roman letters or the phonetic symbol sets of the languages being processed, e.g., *kana* for Japanese or *zhuyin-fuhao* for Chinese. Homophones are resolved by pressing a "look-up" key at the end of each word-unit entry. At that point a support program performs a dictionary look-up and presents the operator with a list of homophone options from which the correct character(s) can be selected.

The disadvantages of this system are that the supporting software is complex and it tends to be slow. When applied to Chinese, there are two other sources of difficulty. First, most Chinese do not compose written material in terms of phonetic characters, and therefore somewhere along the line they must learn to "think phonetically." Secondly, the same phonetic system may not work for all Chinese due to the significant phonetic divergence of the many dialects of spoken Chinese. Nevertheless, as Becker puts it:

> The unavoidable fact is that persons who wish to type Chinese must first take the time to learn something, and in Chinese society their time is far better invested learning standard Mandarin pronunciation than in memorizing some computer company's ad-hoc set of coding-scheme rules. Ref. 3: p. 32

b. BETA: An Automatic Kana-Kanji Translation System (Toshiba JW-10)

This system [cf. Ref. 12] attempts to provide software support to resolve the two basic problems in the processing of Japanese sentences:

- ☞ Segmentation of input into syntactic units (*bunsetsu* in Japanese) for processing, and
- ☞ Resolution of homophones (recall that Japanese has the severest homophone resolution problem among the oriental languages because it has the fewest number of unique syllable pronunciations).

The segmentation problem arises from the fact that Japanese speakers do not naturally build up their sentences from discrete word units. It was noted earlier that Japanese is an agglutinative language. The use of highly synthetic suffixing practices causes a large variation in the forms containing basic semantic information. In short, where one word ends and the next word begins is difficult for the Japanese to pin down. Thus, the Japanese input operator is not accustomed to segmenting each sentence into words and cannot be depended upon to put a "word-end" character[6] in the correct position. He might not place it in the same position as the next operator. The only reasonable solution is for the word processing equipment to accept an unsegmented stream of input

[6]The choice of this character would be wholly arbitrary and system-dependent since Japanese has no segmental symbol to separate words, such as the blank space used in English.

characters and then have the software provide a consistent segmentation internally.

The Beta-system attempts to parse a stream of input symbols into *bunsetsu* (loosely, "phrases"). *Bunsetsu* can be viewed as being composed of an independent part and a dependent part, and each of these two components can be built up according to a regular grammar. Once *bunsetsu* are identified, then, a finite state automaton can support their further analysis. The weak link in the present system is getting the sequence of *bunsetsu* right. The system utilizes an algorithm called "The longest string matching method of two *bunsetsu*." In experiments using the system, it was able to segment *bunsetsu* correctly about 94 percent of the time, and it was able to substitute the correct *kanji* for *kana* symbols about 89 percent of the time. The lower percentage for correct *kanji* substitution is a result of a relatively unsophisticated homophone resolution method in the present system. In cases where the *bunsetsu* and *kanji*-homophone analyses are incorrect, the operator must supply manual correction techniques (editing). Editing of this kind differs from word processing as we know it with English. The English language word processing systems are able to construct a consistent internal representation of the input stream which is 100 percent true to the written language which the operator enters into the computer. Thus, the word processing operator edits only when he is dissatisfied with the meaning of the language itself. With automatic symbol conversion programs, such as *kana-to-kanji*, the operator must do two kinds of editing: first he must correct errors in

the internal representation in the input stream caused by inadequacies in the software translation program. and only then can he proceed with the semantic editing of the sort done in English language word processing.

c. Chinese Hybrid Systems

The Chinese homophone problem differs from that of Japanese and Korean. The basic symbol set of Chinese characters used in word processing is larger: 7000-8000 characters compared to about 2000 used with Japanese or Korean. Thus, there is a potential for more homophones. But the phonology of Chinese provides more possible pronunciations, naturally resolving some homophone conflicts which occur in the other languages. An enduring problem with written Chinese is that the smooth, processed text cannot permit the presence of unresolved phonetic characters. This option is always available with Japanese and Korean. So we might expect homophone resolution software which attempts to be extraordinarily precise in Chinese word processing. Various techniques have been developed to cut down on the size (and cost) of these programs. One such technique, included in the systems described by Sheng, has the operator indicate the tonal category of the syllable along with its phonetic transcription. This simple procedure, costing an extra keystroke per syllable, has the potential to eliminate an average of over 75 percent of incorrect homophones. Another technique is to include some semantic (non-phonetic) information with the input transcription. With the *pinxxiee* word processor, described by

H. C. Tien [Ref. 11: p. 66]. the transcription of a Chinese character is constructed as follows:

hanzi = pinxxiee = pinyin − tone + radical (semantic component)

With the additional keystroke for the radical, incorrect homophones can be virtually 100 percent eliminated. However. the cost is high: there are 214 traditional radicals which must be distributed over a limited number of keys (presumably we do not desire to add 200 new keys to the keyboard). Also. the connection between the semantic clue (radical) and the meaning of the syllable may not be obvious; it may be forgotten or mistaken by the operator. Chinese word processing experience has not yet provided the means to determine if a hybrid coding system like *pinxxiee* is superior to straight phonetic input backed up by complex software.

III. SIZE ESTIMATE FOR A GENERALIZED FONT MEMORY

In the previous chapter it was suggested that computing equipment could be used with greater flexibility and imagination if we improve the capacity for symbol set manipulation. Achieving complete generality is our long-term goal, and we identified two subordinate areas which we want to improve through the present study:

- ☞ Customizing character fonts for a given symbol set to provide a variety of user-specified forms, or type sets, for any particular symbol in the set.
- ☞ Accommodating rapid definition and design of symbol sets containing an arbitrary--perhaps very large--number of symbols.

Figure 3.1 presents a conceptual view of how the overall research effort is organized. The empty boxes in Figure 3.1 represent future contributions to this field. For the present, we offer the BUILDFONT Font Editing System in response to the need for improved tools to support this research and for new applications. The BUILDFONT System is discussed in detail in Chapter 5. In the present chapter, we complete our analysis of the word processing application which was started in Chapter 2. From experience with the systems developed to support the Chinese, Japanese and Korean written languages, we obtain an estimate of the size of "ideal" arbitrarily bounded symbol sets.

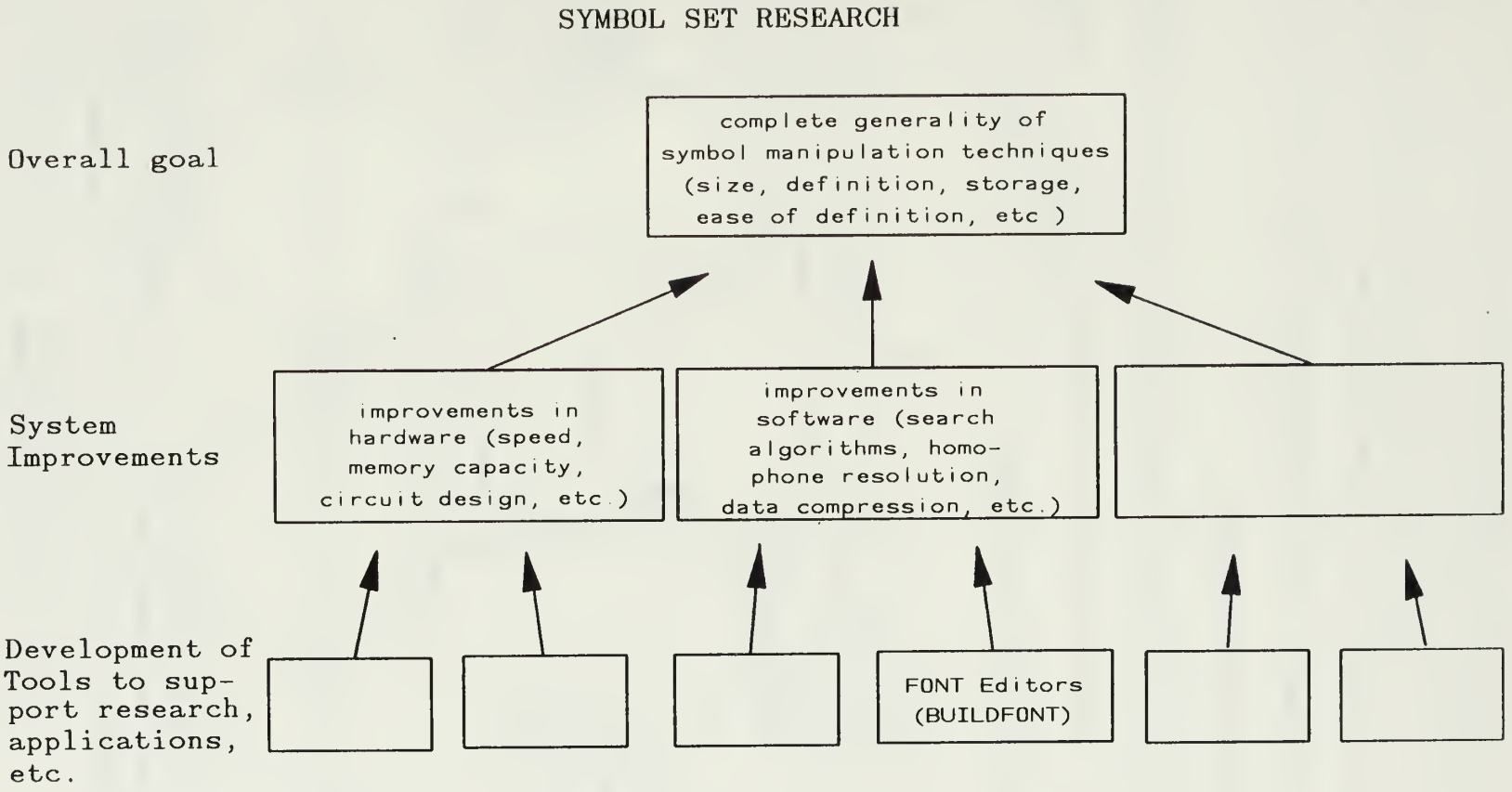


Figure 3.1
Support for Improved Symbol Set Manipulation

We began describing the scope of the general word processing problem by examining the linguistics of Chinese and Japanese. to see how these languages use extremely large character sets to express their respective written languages. We then went on to present a summary of the word processing systems that attempt to cope with these written languages.

Figures 3.2 and 3.3 illustrate word processing systems for the languages we have discussed. Figure 3.2 represents the hardware and software components for a typical English language word processor.

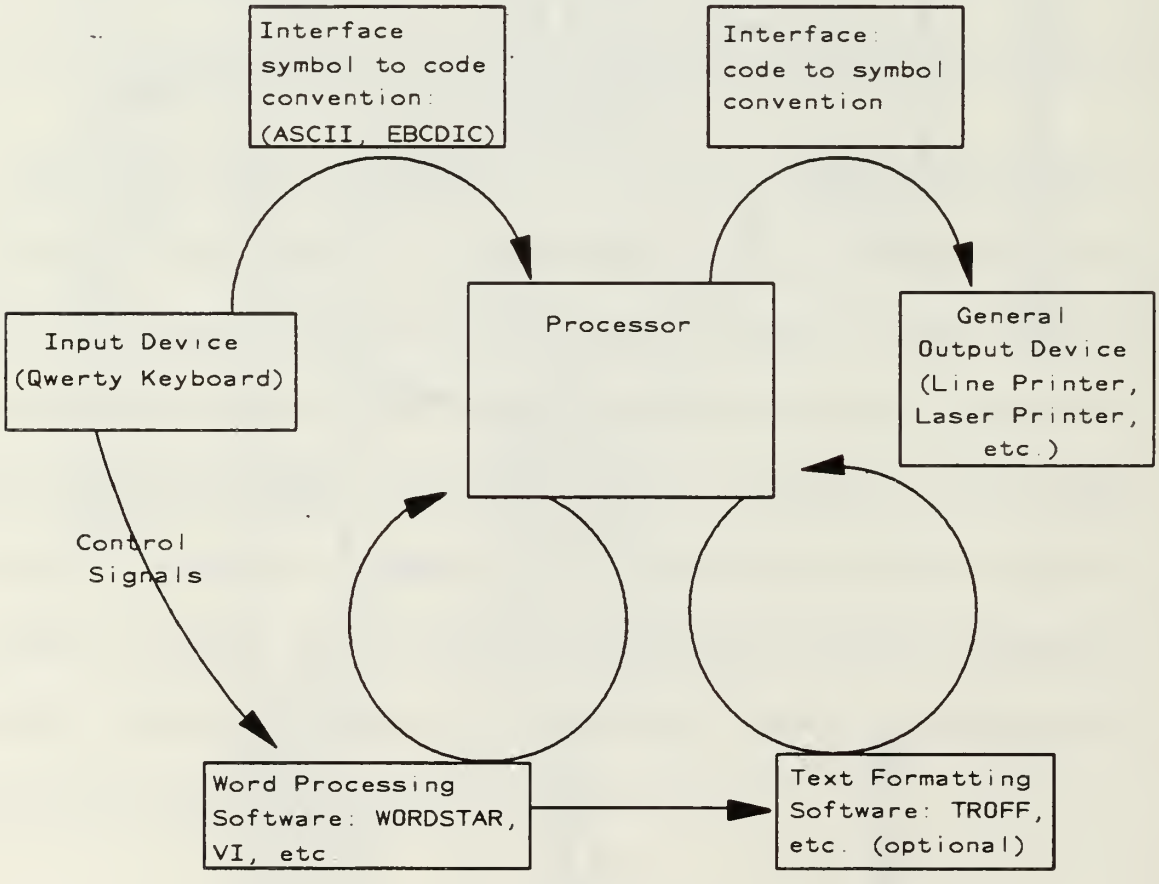


Figure 3.2
Typical English Language Word Processor

Figure 3.3 depicts a general word processor which must handle the large character sets required by Chinese. Japanese and Korean.

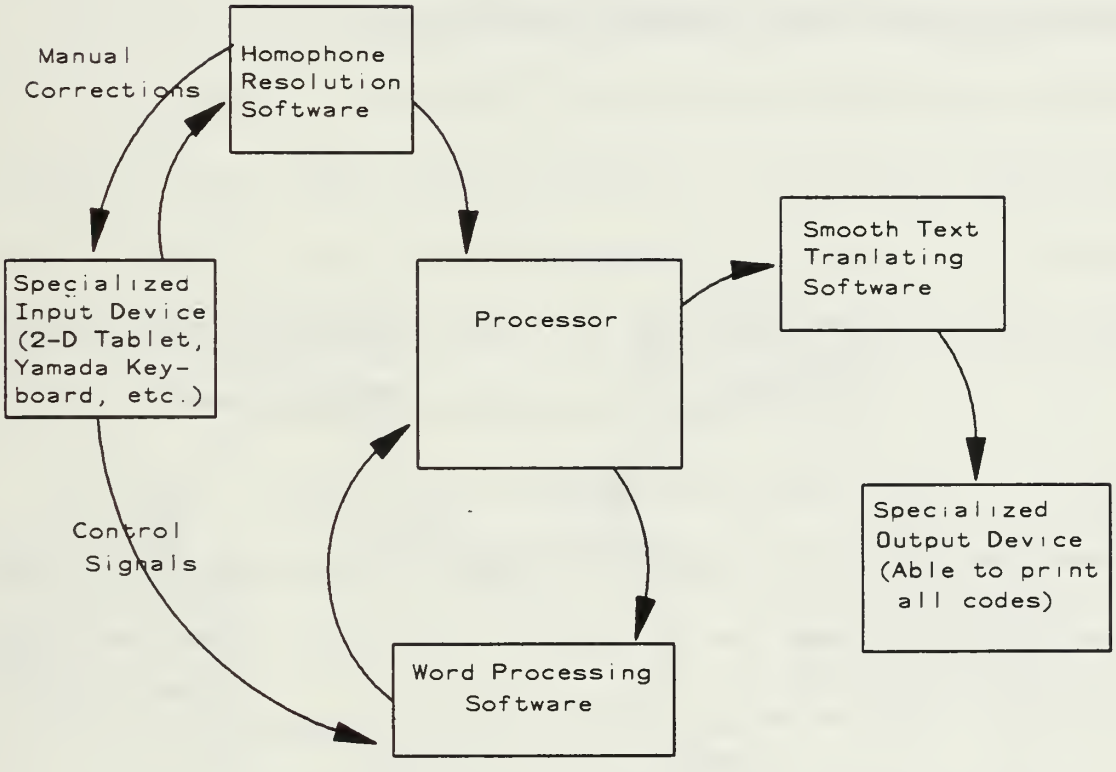


Figure 3.3
Present Oriental Language Word Processing Systems

Figure 3.4 illustrates the "Ideal" Word Processor, a system which can accommodate any symbol set the user may desire. Note that the system represented in Figure 3.4 is very similar to that of Figure 3.2. This suggests that a computing system capable of accommodating arbitrarily large symbol sets can reduce the entire word processing problem for written languages such as Chinese to the proportions of English language word processing.

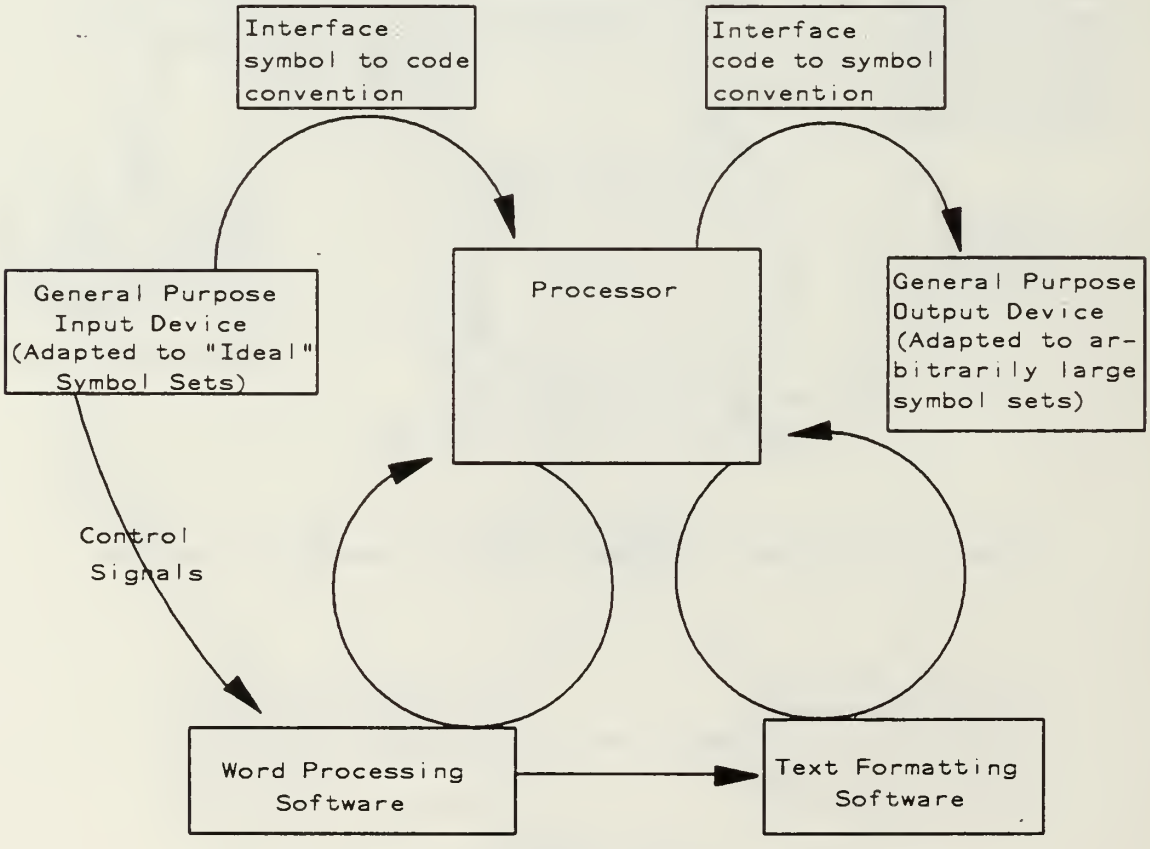


Figure 3.4
The "Ideal" Word Processing System

In Chapter 2 we stated that the word processing application best represents the dimensions of the problems we are trying so solve as we improve overall symbol manipulation generality. These dimensions are measured in terms of symbol set size and symbol complexity.

A. THE SIZE DIMENSION

It has been noted that the ASCII and EBCDIC character sets provide codes for up to 128 (ASCII) or 256 (EBCDIC)[7] characters and control signals, and that members of either of these sets can be coded in a single eight-bit byte. Since ASCII or EBCDIC coding conventions pervade the present level of western language word processing technology, it is reasonable to use 128 as the lower bound for the size of a word processing character set.

What about an upper bound for these character sets? In theory we would like to be able to process character sets of limitless size, but for designing practical systems we need an upper limit. Suppose we were going to design a word processing system and we wanted the capability to process English and Chinese text, including every possible written character. Table 3.1 provides and estimate for our maximum size character set.

[7]Since EBCDIC codes are eight bits long, "256" is actually the capacity to store discrete codes ($2^8 = 256$). EBCDIC employs a "binary-coded decimal" scheme, so only a fraction (~ 194) of the codes available are used.

TABLE 3.1--MAXIMUM SIZE FONT	
Type of Symbol	Number required
English alphanumeric characters, special characters (punctuation, etc.), and control characters	128
Traditional Chinese characters	≃ 50,000
Simplified versions of the traditional forms (used mainly in the People's Republic of China)	≃ 5,000
Chinese phonetic characters	37
Traditional "radicals" (character components used for dictionary classification)	214
Simplified "radicals" (used in the PRC, and for non-phonetic coding used in some word processing systems: TCCM and others)	≃ 400
Subtotal	≃ 55,779
Capability for Japanese language processing	add ≃ 1,100
Capability for Korean language processing	add ≃ 400
Total	≃ 57,279

If we consider 58,000 to be our maximum size character font for a general Oriental language/English language word processing application, then we see that we must be able to store 16-bit codes in order to reference individual symbols in the font table ($\log_2$ 58,000 = at least 16). Three items of note are the following:

- The sheer size of the character set forces abandonment of 8-bit ASCII and EBCDIC coding in favor of 16-bit codes.

- Inclusion of the English language character set really costs very little, once we accept 16-bit codes to facilitate indexing of the 55,000+ symbols needed just for Chinese. We automatically obtain a font table which can include up to 64K discrete symbols. Thus, we are free to "throw in" a few hundred extra symbols if we want to, so long as the total does not exceed 64K. For languages such as English, where the total number of symbols needed is some two orders of magnitude less than that for Chinese, we see that it is almost insignificant to the hardware if we include the additional symbols. However, it is of great practical importance to potential users to be able to process many languages on the same equipment.

- 58,000 symbols is a truly large set. It is hard to imagine an application requiring a larger number of symbols. This is why we have chosen the word processing application to represent the greatest dimensions of the symbol manipulation problem.

B. THE COMPLEXITY DIMENSION

Now that we have determined the maximum number of entries in a font table[8], we need to estimate the memory required to store it. Let us assume that each symbol is stored as a bitmap. Matsuda has collected data on dot matrix printing of Japanese characters which indicates a 24x24 matrix is required for acceptable quality [Ref. 5: p. 43]. With the use of a compound dot matrix

[8]The role of "font tables" in symbol set management is discussed in Chapter 4.

method. good quality can be achieved with a 16x18 dot matrix size. For high quality characters Yajima. et.al.. state that resolutions of 64x64 or better are required [Ref. 14: p. 222]. However, we use 24x24 for our upper limit estimate.[9]

If the entire bitmap is stored, each entry in the font table would require 72 8-bit bytes. We assume that we only need to store a portion of each bitmap, a rectangle containing all actual points used to form the symbol, and we estimate that the "average" Chinese character requires storage of eighty percent of the bitmap, or about 60 bytes. Thus, the maximum storage requirement is (64K X 60 bytes per character) = 3.84 megabytes of font memory for one character font of the kind we described in this section.[10]

C. MID-RANGE FONT TABLES

Fortunately, no one really needs all 50,000 Chinese characters to do adequate word processing. About 2000 characters account for 97 percent of the symbol usage. Adding 1000 more covers 99+ percent of usage. With 4000-8000 characters, virtually 100 percent of all ordinary word processing needs are satisfied. Japanese and Korean systems can get along quite comfortably with

[9]This is considerably higher resolution than the standard 9x9 bitmaps for font elements in the Silicon Graphics, Inc., Iris-2400. The greater complexity of Chinese characters (including those adopted by the Japanese and Korean languages) necessitates this increase to a minimum of at least 24x24 bitmaps. The "acceptable quality" of these characters should be sufficient for the word processing interactive display. Better resolution for fine quality printing could be made available in the font capacity of the printing device, but this need not affect our estimate for the interactive display.

[10]This figure assumes a font storage that is (8-bit) byte allocatable, which may not be the case for all equipment configurations. For example, on the Silicon Graphics, Inc., Iris-2400, which is the host system for the BUILDFONT Font Editor, font memory is allocated in 16-bit chunks (minimum).

2000-2500 characters.[11] In Table 3.2. we show revised size estimates for storing a practical font table (revised to mid-range values consistent with the symbol set requirements described above).

TABLE 3.2--MID-RANGE FONT MEMORY SIZES		
FONT SIZE	FONT MEMORY REQUIRED	
8000 char	(8K X 60 bytes per char)	= 480 Kbytes
6500 char	(6.5K X 60	) = 390 Kbytes
3000 char	(3K X 60	) = 180 Kbytes
2500 char	(2.5K X 60	) = 150 Kbytes
2000 char	(2K X 60	) = 120 Kbytes

D. A PRACTICAL SYSTEM

It can be seen that about 500 Kbytes of font memory is sufficient to provide a fairly comprehensive character set for Oriental language word processing, including several styles of Roman character fonts, Japanese *kana* and Korean *hangul*.[12] By limiting the number of Chinese characters to 3000 or 2500, more than one style of Chinese character may be included as well. If a designer desires to remain within a limit of 500 Kbytes of font memory, a word processing system supported by this resource should include a font editor utility, such as the BUILDFONT System, to generate a new symbol on-the-spot whenever the user needs an obscure or archaic character intentionally omitted from standard fonts.

[11]Glossary entries provide numbers of symbols contained in some of the standard character sets mentioned in the first section of Chapter 2. The largest standard set is China's "Standard Cable Code" with 8085 symbols.

[12]Once again, hardware specific considerations noted in footnote 10 must be kept in mind.

IV. GRAPHICS SUPPORT FOR SYMBOL MANAGEMENT SYSTEMS

A. THE IDEAL WORD PROCESSOR IS A VIRTUAL COMPUTER

In Figure 3.1 we set achievement of "complete generality of symbol manipulation techniques" as the goal for our long-term research efforts. In subjective terms, we will have reached our destination when we have created a *virtual computer*[13] which will accept any stream of symbols we choose to feed into it. This virtual computer must then be able to interpret the input symbols by either displaying the correct graphic representations (as in "echoing" the input), or by performing some other intended computation in response to signals derived from the symbols.

In Chapter 3 we became more objective in our analysis by applying the experience of the word processing application with non-Roman symbol sets to estimate how much font memory we need for this "Ideal" Word Processor. Since the Ideal Word Processor was singled out as an application spanning the largest dimensions of the symbol manipulation problem, then it follows that we will have created the virtual machine we seek if we can design a computing system to host the Ideal Word Processor. Any less ambitious application will automatically be manageable within this capability.

[13]We use the term "virtual computer" to represent the idea that once we have designed an actual hardware system that can support our most ambitious symbol manipulation applications, the fact that this system has physical limitations is concealed from the user, and the system appears to have virtually unlimited capacity.

At this point, it is evident that the perfection of generality of symbol manipulation techniques does not depend on a revolution in the design of computer architectures. In Chapter 3 we arrived at a total of either 3.84 Mbytes or 500 Kbytes for the amount of font memory needed for a workable Ideal Word Processor. By present day standards this is very large, but it is not prohibitively large. We have not yet analyzed the degree to which font memory access speed must be improved, if at all. Let us make the assumption that, as we have found with font memory size, the required improvement in access speed will occur as a by-product of advances in more technologically challenging areas of computer architecture and circuit design. We justify this assumption in the following sections.

B. COMPUTER GRAPHICS DEVELOPMENT VIEWED FROM THE FONT MANAGEMENT ASPECT

Evolutionary improvement of computer symbol management capability has, in fact, occurred as an ingredient within the development of better and better interactive display systems. This makes the design of abstract symbol management systems and the provision for hardware font memory a research area within the wider realm of computer graphics. Zyda [Ref. 15] touches upon historical aspects of computer graphics in his summary of the phases, or cycles, in the development of the modern graphics workstation. In this view, the present generation of "leading-edge" graphics workstations is the product of a design philosophy which is in the third of a series of developmental cycles.

1. The First Cycle

This phase began 15-25 years ago with efforts to develop a capability to perform real-time, interactive applications (simple ones, at least). At that time the objective was to move from a batch-processing, card-reading environment toward single-user systems which could display symbols and pictures. When a primitive capability to do this was realized (by the adaptation of direct keyboard input devices and CRT displays as output devices), the performance of these systems was then evaluated. New hardware components were designed and developed for the purpose of speeding up the delivery of graphics objects to the display, i.e., perfecting the "real-time" response demanded by system users.

2. The Second Cycle

This phase corresponds roughly to the evolution of super minicomputers during the mid-to-late-seventies. Researchers within the computer graphics area continued to respond to the upward spiral of user performance demands by developing hardware solutions for applications that had been impossible previously. For example, matrix multiplier circuits facilitated the real-time performance of linear algebra for scaling, rotating and translating graphics objects. However, concurrent with the enhancement of pure graphics support capabilities, the technology of general-purpose hardware for the single-user computers hosting the graphics systems was also improving. Early minicomputers (such as the DEC PDP-11 series) gave way to super minicomputers like the DEC VAX-11 series). These more-powerful machines

provided enhanced, general purpose features such as virtual memory, time-sharing and multi-tasking, which reduced their capacity for dedicated support of graphics applications.

3. The Third Cycle

The culmination of the current phase of computer graphics improvements is the contemporary graphics workstation, which represents efforts of the eighties to move back toward dedicated support for single-user graphics applications. The system design features which have led to the workstation concept focus upon inclusion of a separate, dedicated processor (Display Processing Unit--"DPU") responsible for control of the graphics-specific operations within the host computing system. A very general view of the configuration of such a system is presented in Figure 4.1 [Ref. 15: fig. 1].

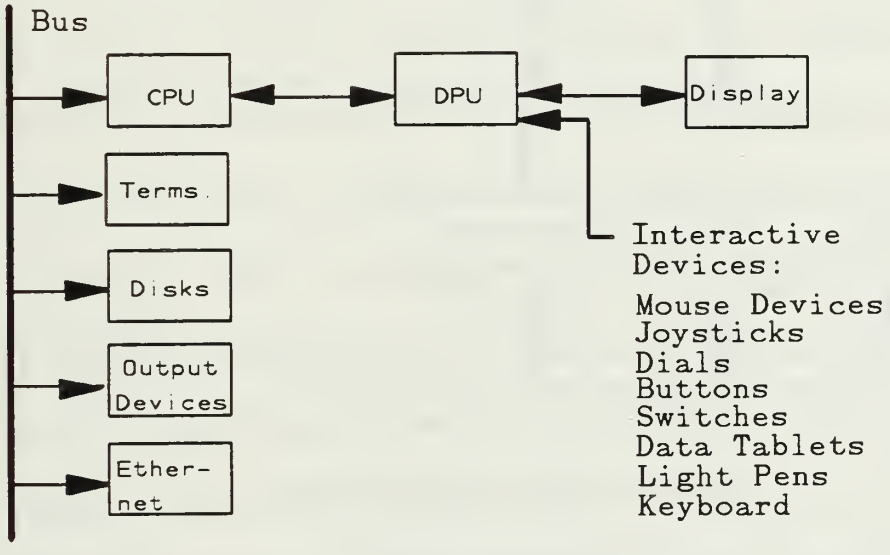
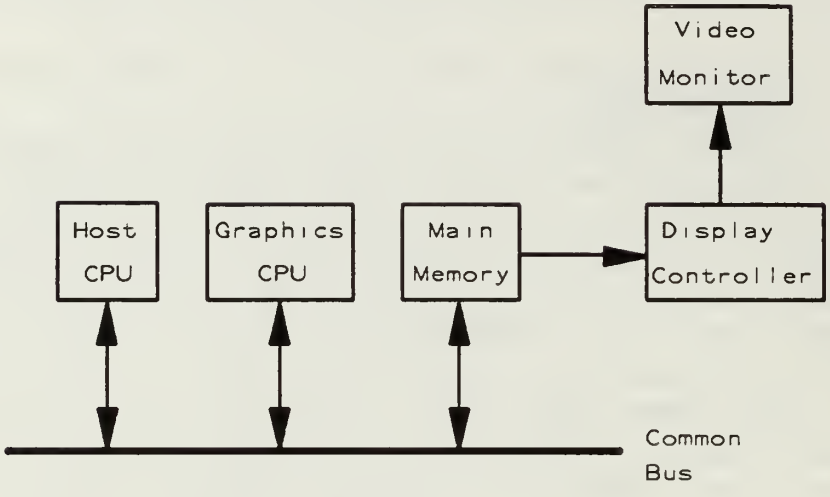
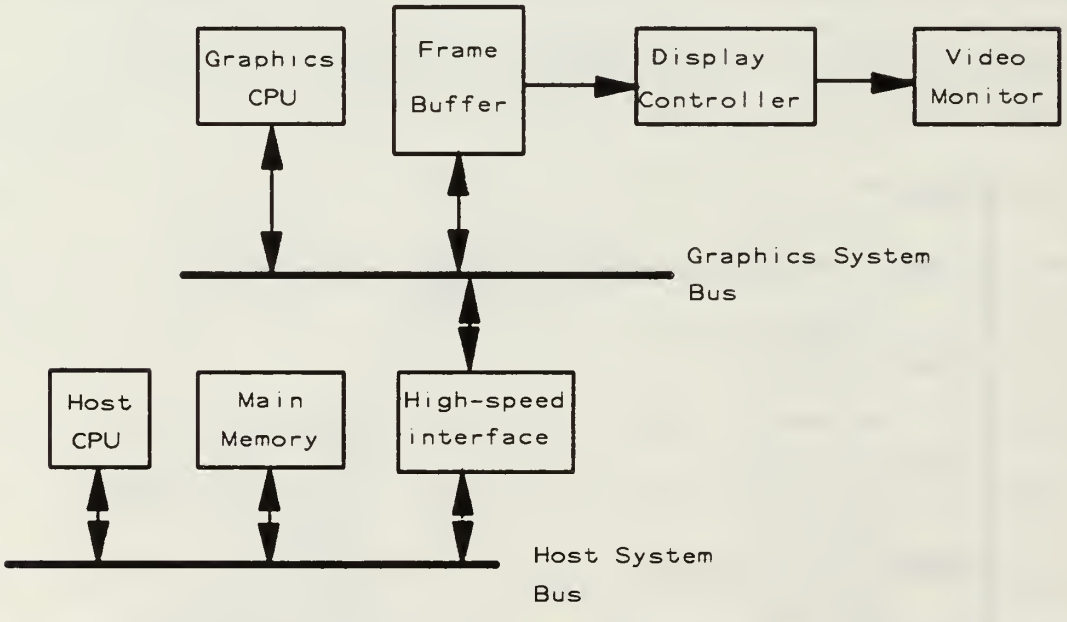


Figure 4.1
Block Diagram of Interactive Graphics Workstation

Figure 4.2 [Ref. 16: p. 63] presents variations of how the graphics subsystem may be organized within the overall workstation format of figure 4.1.



(a) Common Bus Architecture



(b) Frame Buffer Architecture

Figure 4.2
Graphics Workstation Architecture Variants

4. The Leading-Edge Graphics Workstation

The maturation of third cycle research efforts is exemplified by today's "leading-edge" graphics systems, including the Silicon Graphics, Inc., IRIS-2400, selected as the host system for the BUILDFONT Font Editing System. A block diagram of the IRIS System architecture is presented in Figure 4.3 [see also Ref. 15: pp. 8-10 and Ref. 17], and a specification summary is included in Appendix B. The IRIS System utilizes the raster refresh display technique [cf. Ref. 16: pp. 8-19], whereby all text and picture objects to be displayed are processed through a pipeline of specialized circuitry until they are finally deposited in the "frame buffer" in the form of (a two dimensional array of) "pixels." The refresh subsystem functions to project, or map onto the screen by electronic means, the pixel arrangement stored in the frame buffer.

5. Handling of Text By the IRIS System

Up to now we have used the term "font memory" without elaboration to loosely refer to some location within the computing system where we store the data needed to produce textual characters and symbols on the display. Let us now define "font memory" more precisely by saying that it is a dedicated hardware computer memory set aside to contain character data which can be mapped directly into a desired position in the frame buffer. In a raster refresh graphics system such as the IRIS System, establishing the font memory and its data path to the frame buffer solves the problem of getting text onto the display. But the proper data must be placed in font memory to begin with.

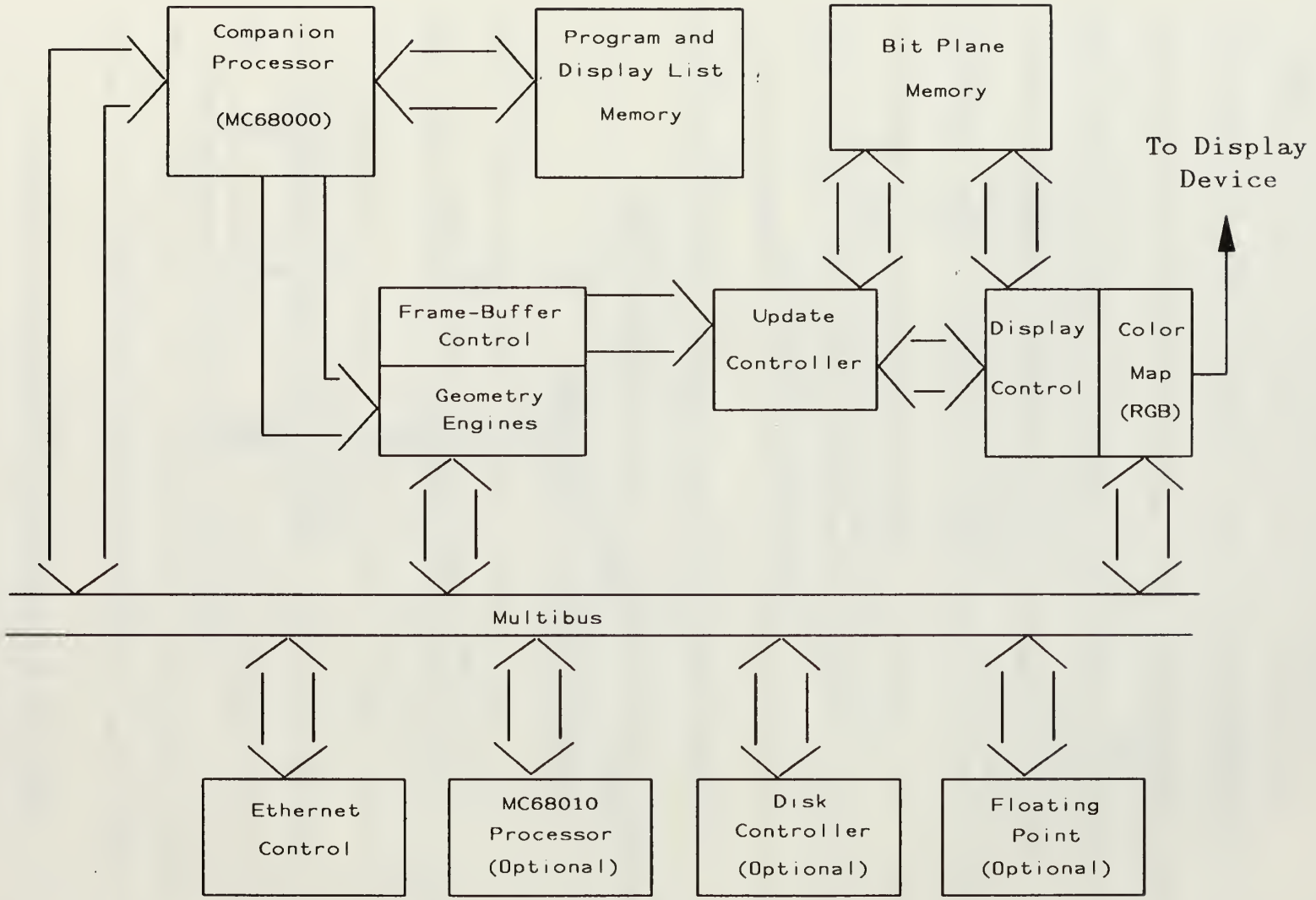


Figure 4.3
Block Diagram of IRIS System Architecture

before graphics text can be synthesized and moved to the frame buffer. The IRIS System provides a "default" font of 9x9 pixel characters which are stored in permanent font memory (ROM). The system also provides 16 Kbytes of RAM font memory where the user can load other fonts of his choice.

Figure 4.4 illustrates how IRIS font memory is accessed and how the system performs data conversions needed to support font management. Customized character sets which can be placed in font memory by the user are normally kept as formatted font files (example with explanation shown in Figure 4.5) in external disk storage. High level software routines (programs) are needed to bring a font file into main (RAM) memory where the data are stored in data structures. the *font table* and the *raster array* (explained in detail in Chapter 5). The IRIS System utilizes the ASCII coding convention, and thus a font file may contain data for up to 128 characters. As prescribed by the file format, each character in the file has a line of parameter data followed by bitmap data from which the system will eventually determine the size and form of the character. When the file is read into RAM memory, each character's parameter line is stored as a *font table* entry, and the bitmap is read into the *raster array* (where the two-dimensional bitmaps are strung out linearly, one data word after another). The *raster array* can then be loaded into font memory by means of the system function *defrasterfont* [Ref. 18: pp. 5-6 and 5-7]. The linearly arranged bitmap data placed in font memory together with parameter values retained in the *font table* enable the system to accurately reconstruct the correct pixel bitmaps in the

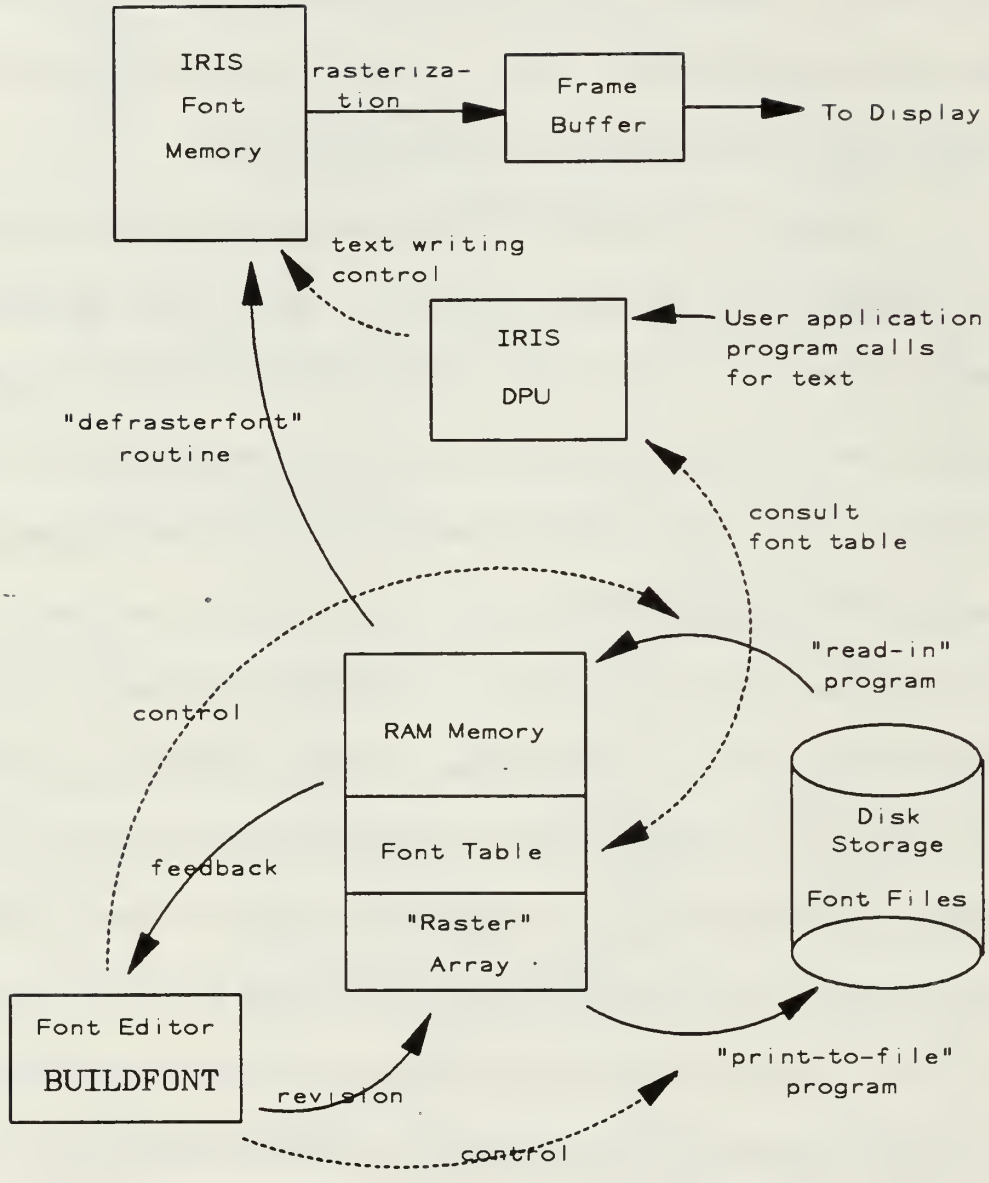


Figure 4.4
IRIS System Font Management Data Conversions

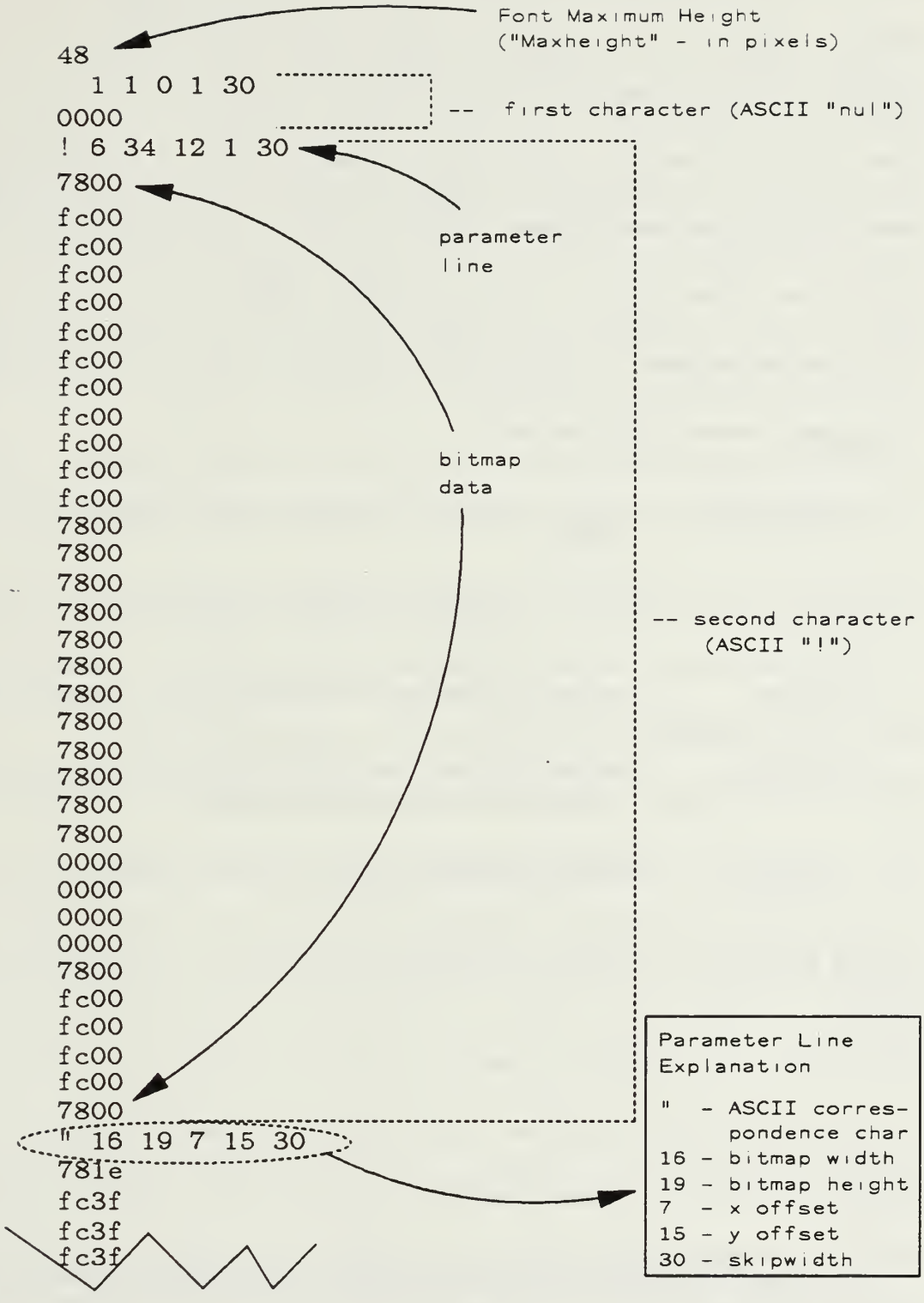


Figure 4.5
Example of a IRIS Formatted Font File

frame buffer. To create a line of text when programming the system to produce a graphics object. the user must first designate the desired font with the *font* function (or, if he chooses not to do this. he gets the default font). Next, the user must specify the position on the display projection where he wants the text to start by calling the *cmov* function. Finally, the user must designate the string of symbols he desires with the *charstr* function [Ref. 18: p. 5-7]. For other programming purposes the system offers users the following "built-in" functions supporting character manipulation[14]:

- ☞ defrasterfont -- loads character data from main memory into font memory.
- ☞ font -- selects the desired font (in the font memory).
- ☞ getfont -- returns the number designating the font currently in use.
- ☞ getheight -- returns the maximum height value of the font currently in use (the value is a number of pixels).
- ☞ strwidth -- returns the width (in pixels) of a text string.

System primitives to allocate and deallocate font memory, etc., are invisible to users.

[14]The names of functions listed are those of the "C" programming language. The IRIS System also supports FORTRAN and Pascal, and the same functions are available with slightly variant names.

6. The Fourth Cycle

It has been stated that the objective of the third cycle of improvements to computer graphics has been to remove graphics support functions from the general purpose computing environment and concentrate them into "workstation organization" dedicated to support of single-user applications. This line of development has thus spawned the appearance of today's "leading edge" graphics workstations, including the IRIS System described in the previous sections. Progress within this third cycle philosophy will continue, resulting in even more powerful workstations for the near future. For example, Silicon Graphics, Inc., has already designed a successor to the IRIS-2400 which promises greatly enhanced capability. This system is scheduled for release about 1988.[15]

Third cycle systems, however, address the traditional problem areas of computer graphics design--enhancing general graphics (hardware) functionality in response to user demand for more efficient processing of applications. Zyda [Ref. 15: pp. 11-12] sees limits to the extent to which these enhancements can continue, within the third cycle approach. However, the eventual reaching of these limits is not expected to diminish the user's appetite for better performance. A new development phase, the fourth cycle of computer graphics improvements is logically inevitable. This phase may be characterized by efforts to design the applications themselves (i.e., the algorithms) into hardware circuitry.

[15]Information on this system was presented by J. H. Clark during a briefing at the Naval Postgraduate School, Monterey, California, on 14 May 1986.

When we discuss the idea of rendering algorithms into hardware circuitry, we clearly arrive at a point where the present technology can support generalized symbol manipulation techniques described in the first three chapters of this study. With these particular applications (symbol manipulation systems), we cannot differentiate between designs having efficient symbol manipulation capability of a completely general nature (third cycle workstations) and designs created with specific applications in mind (fourth cycle application-specific architectures). This is because development of symbol manipulation capability, even within the field of computer graphics, has lagged behind the more challenging design aspects (vector processors, matrix multipliers, etc.) which have received emphasis. We see in Figure 4.3, great sophistication in the circuitry devoted to the general "graphics pipeline." The conspicuous exception is font management support. As a result, we were able to summarize font management-specific functions of the IRIS System as the relatively meager collection (5 total) listed in the previous section. Thus, the third cycle has hardly run its course with respect to symbol manipulation improvement. This fact will not delay the approach of the fourth cycle, however.

It has been noted that research in computer graphics design has always reponded to ever-unsatisfied user demands for better systems. The capability to project text onto a video screen was one of the original user demands, and ironically, not much more has been demanded since that capability was attained. This study attempts to rekindle the demand, and in so doing, take

a step toward designing a better balance of capabilities into the next generation of graphics workstations.

V. THE BUILDFONT SYSTEM

In the first three chapters of this study we identified the need and discussed the motivation for improving computer-based symbol management systems. We listed applications that become possible with the perfection of completely general symbol manipulation techniques. We went into detail with one application area: Word Processing of written languages which employ non-standard character sets. We reviewed some of the problems caused by limited symbol manipulation generality in the present word processing systems developed for these languages, and we proposed an "Ideal Word Processor" to eliminate these problems. In Chapter 3 we estimated the maximum size font memory needed to support a general-purpose symbol manipulation feature (in this case, the Ideal Word Processor).

In Chapter 4 we placed the task of improving symbol management systems within the overall scope of computer graphics research, since the most innovative symbol manipulating systems have traditionally appeared as specialized graphics support features prior to becoming standardized features of general purpose computing installations. It was noted that user demand for better performance of application programs has traditionally provided the impetus for the cycles of hardware development within the computer graphics field. Taking the IRIS-2400

Graphics Workstation as being representative of "leading edge" graphics capability. we have developed the BUILDFONT Font Creation and Editing System, a software development tool to facilitate the creation and maintenance of customized symbol fonts used in IRIS System applications. BUILDFONT, then, is a tool which, by facilitating and expediting the creation of customized fonts, serves to support development of the IRIS System applications critical to symbol management research.

A. HOW THE BUILDFONT SYSTEM WORKS

In the last chapter, we described the features of IRIS System font management. and we observed that certain data conversions must take place to use a *special* font (i.e., other than the *default* font). Special fonts are stored as font files in secondary (disk) storage until they are needed by an application. A font file contains two kinds of information for each symbol in the font: 1) bitmap data which define the actual size and shape of the symbol. and 2) the symbol "parameter line," which stores information *about* the symbol. When an application needs to use a special font. the font file is read into RAM memory, where the data for each symbol are divided and stored into two data structures: the *raster array* (for bitmap data) and the *font table* (for the parameter line)(refer to Figures 4.4 and 4.5).

The BUILDFONT System is itself an application program which uses special fonts: It allows the user to create new ones or to change existing ones.

BUILDFONT creates a special font by establishing the font table and the raster array and then by depositing new data into them. The user controls the flow of this data interactively. BUILDFONT edits a font by reading the appropriate font file into the raster array and the font table, and then interactively replacing original data with the desired updates. The BUILDFONT System can be evaluated for its functionality and user-friendliness by the number, convenience and efficiency of the operations it provides in accessing the font management data structures. As these data structures are modified, the BUILDFONT System provides feedback by loading the raster array into font memory (via the *defrasterfont* function) and displaying the updated font symbols to the user in the manner of any other IRIS System application.

B. CHANGING THE FONT MANAGEMENT DATA STRUCTURES

Conceptually, each font symbol used by the IRIS System is defined by its bitmap, a two-dimensional array of pixels. The rectangular perimeter surrounding the bitmap is known as the symbol's *bounding box*. In the IRIS System, symbols are displayed by specifying a reference location (in 2-space) called the *current character position*. The bounding box containing the symbol is then placed in the frame buffer at a location relative to the current character position. The status of pixels contained in the bounding box is determined by the data stored in the *raster array*. The relative position of the bounding box (with respect to the current character position) is determined by the *font table*

parameters. Figure 5.1 shows how the information stored in the font management data structures is interpreted.

1. Changes to the Raster Array

A modification, addition or deletion to the data in the *raster array* results in a change to the *size* and *shape* of a symbol in the font. However, this action does not affect the symbol's location relative to the current character position.

2. Changes to the Font Table

Modifications to the parameters stored in the *font table* result in a change of the *location* where the symbol is displayed relative to the IRIS System's current character position. However, this action has no effect on the size and shape of the character.

3. Precise Description of the Font Management Data Structures

The data structures we have been discussing are simply convenient. (IRIS) system-defined abstractions for the blocks of RAM memory where the BUILDFONT System stores data words defining the font symbols. With the bitmap data, BUILDFONT needs many consecutive memory locations to store the data linearly. The bitmaps are broken into 16-bit words (stored conveniently in the disk font file as unsigned hexadecimal integers). So a one-dimensional array is the appropriate data structure. It must be large enough to hold all words from all bitmaps in a font, but it need not be larger than the 16-Kbyte hardware limitation placed on addressable font memory (an IRIS System design constraint).

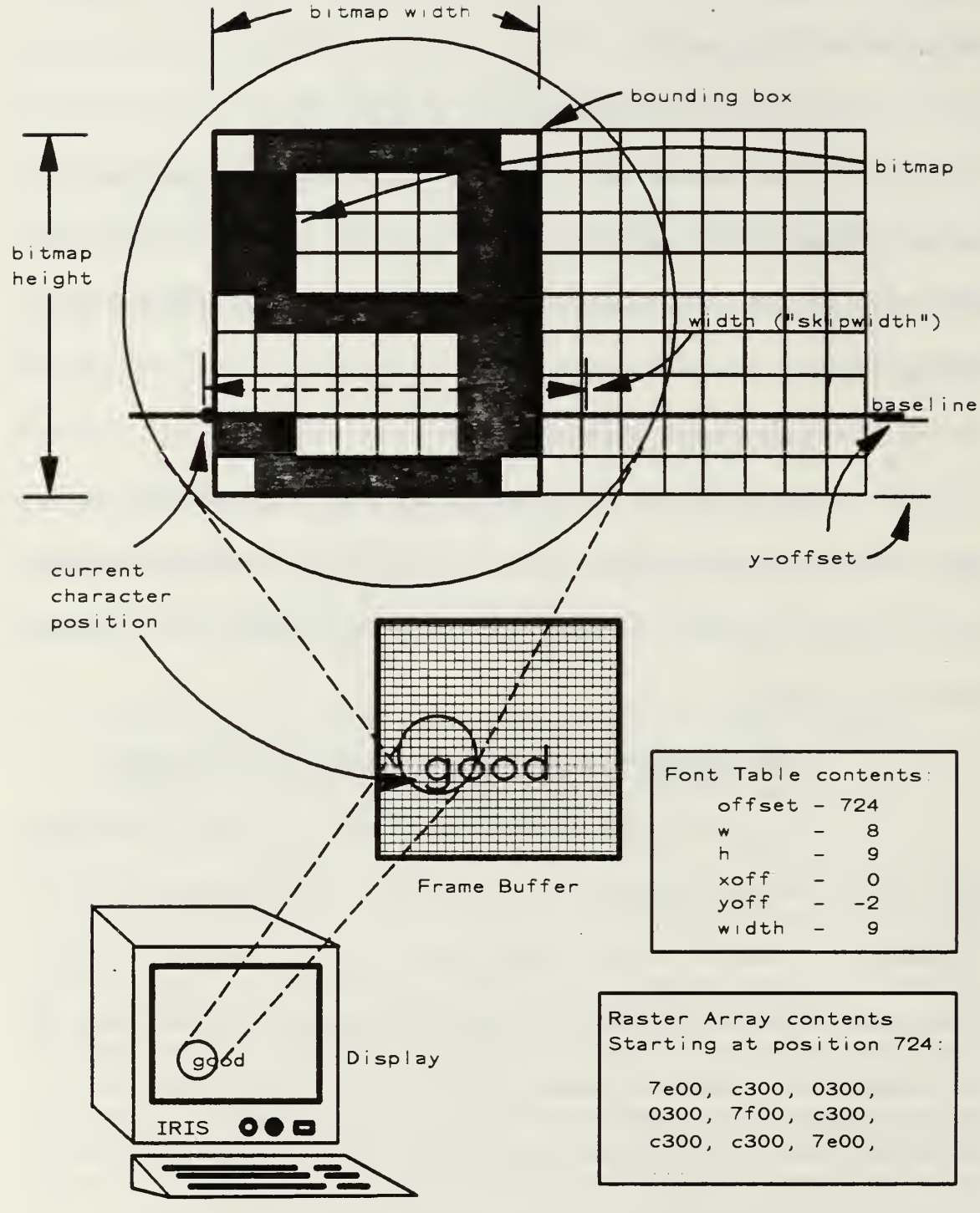


Figure 5.1
Interpretation of Data in Font Management Data Structures

The font table, on the other hand, requires storage of tabular data (the parameter line) for each character in the font. Thus, an array of "record-like" structures is reasonable.[16] The BUILDFONT System is implemented in the C Programming Language, and therefore we declare the *raster array* to be an array of 16K short integers. The *font table* is an IRIS System array of C *structures* [cf. Ref. 19], each containing the following fields (which correspond to a symbol's parameter line) (see Figure 6.1):

- ☞ offset -- the first location (index) in the raster array where the stored bitmap data for this symbol begins. This value is calculated when a font file is read into RAM memory, and so it does not appear in the symbol's parameter line in the font file.

- ☞ w -- the width of the bounding box (bitmap) in pixels.

- ☞ h -- the height of the bounding box in pixels.

- ☞ xoff -- the number of pixels that the bounding box is "offset" from the current character position in the horizontal direction. With zero x-offset, the left edge of the bounding box is even with the current character position. Negative x-offset moves the box to the left; positive x-offset moves the box to the right.

- ☞ yoff -- the number of pixels that the bounding box is "offset" in the vertical direction. With zero y-offset, the bottom of the bounding box is even with the current character position, and the symbol is said to be positioned on the *baseline*. Negative

[16] With the current implementation of ASCII fonts in the IRIS System, all fonts contain 128 characters (smaller fonts are possible by leaving some symbols undefined, i.e., providing no data defining them in the font file). In future implementations which may allow fonts to contain an arbitrarily large number of characters, the font table data structure may require revision (to a linked list or tree organization for the record items, rather than an array.) This will undoubtedly depend upon whether search time efficiency or memory space efficiency becomes the dominant design consideration for future symbol management systems.

y-offset moves the bottom of the box a corresponding number of pixels below the baseline: positive y-offset raises the box above the baseline.

- width -- this value is the *skipwidth* or *x-increment*, the number of pixels the current character position is moved to the right after a symbol is placed in the frame buffer. This value determines the gap that is left between successively printed symbol bitmaps.

With the parameters **w** and **h**, the number or data words defining the bitmap can be calculated. The formula[17] is

$$\text{number of data words required} = [\ (\ w \text{ div } 16\ ) + 1\ ] * h$$

From Figure 5.1, we see that nine consecutive data words (starting at position "724" in the *raster array*) store the data defining the symbol "g".

C. ORGANIZATION AND FEATURES OF THE BUILDFONT SYSTEM

The BUILDFONT System is an interactive, menu-driven program implemented in the C Programming Language and accessible from the UNIX[18] operating system environment of the IRIS-2400 Graphics Workstation. The BUILDFONT utility provides three main capabilities relating to font management:

- Creating a new font and adding symbols to it.
- Editing the symbols of an existing font, including additions and deletions of whole characters.

[17] "div" means "integer divide operation" (discard the remainder).

[18] UNIX is a trademark of Bell Laboratories.

☞ Simply displaying the contents of a font to see what items it contains (this feature is useful in selecting a font from possible font files to support a particular application).

Figure 5.2 is a user's view of functional relationships among the components of the BUILDFONT System. The blocks in the figure are similar in appearance to the actual screen layouts used in the BUILDFONT System. An overview of system operation is presented in the next sections.

An attempt has been made to modularize the functions of the BUILDFONT System and organize the programming modules hierarchically. Thus, the capabilities listed above are accessed through the "Main Menu Level." Once the user reaches this level, he is presented with choices which lead him into the desired area of the system.

1. Creating a New Font

To proceed from the Main Menu Level into one of the functional areas (i.e., "CREATE," "EDIT," or "DISPLAY," but not "HELP" or "EXIT"), the user is prompted to supply a name for the font file on which he intends to work. For editing or displaying an existing font, BUILDFONT uses the name supplied to search for a font file in the user's directory. For creating a new font, BUILDFONT expects a new file name. Once this is received, the font management data structures are initialized to receive the new font-defining data.

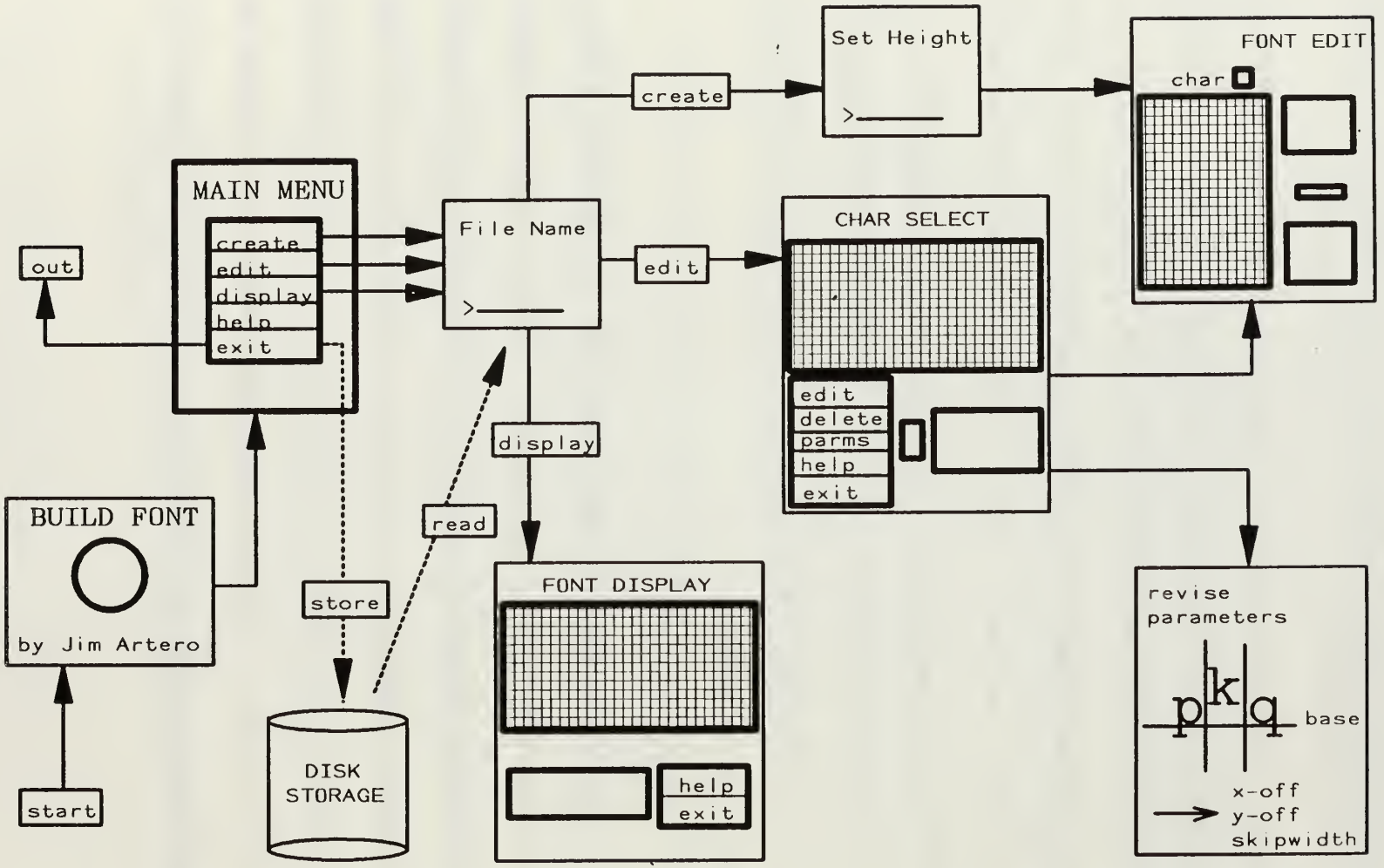


Figure 5.2
The BUILDFONT Font Creation and Editing System

Initialization consists of clearing the *raster array* and the *font table*, and establishing some parameter settings through communication with the user:

- ☞ font maximum height -- the user is asked to select a value for the greatest height of any bitmap in the font. This is the first entry placed in a font file (see Figure 4.5).[19] The maximum height is set only at initialization of the font creation process.

- ☞ bitmap height and width -- once the maximum height is recorded, the user is asked to provide an *average bitmap height*, with which each of the bitmaps in the font will be initialized. The same values are used for the *average bitmap width*, so that the user is given a square grid (representing an empty bitmap) to work with in forming the new symbol. The height and width of the grid can be any value desired as long as they do not exceed the maximum height of the font.

- ☞ default values -- the user may elect to use the system default values for *maximum height* and *average height*.[20]

Once the initial parameters have been set, the program moves into the "FONT EDIT" module, where interactive editing of the bitmap is performed. In creating a new font, the program allows the user to move into and out of the

[19]With an operational font, the physical interpretation of the maximum height value is the vertical distance skipped between lines of text. For example, in graphics programming on the IRIS System, the maximum height value would be used in calls to the *cmov* function to decrement the vertical coordinate of the *current character position* down the screen (page) in applications such as a word processor (e.g., "cmov2i(x, y – maxheight)").

[20]Default values for the current version of the BUILDFONT System are listed in system documentation, available at the Naval Postgraduate School's Graphics and Video Laboratory.

bitmap editing environment until a maximum of 128 symbols have been created (capacity of an ASCII font).

2. Operation of the Bitmap Editor (FONT EDIT)

The FONT EDIT module allows interactive setting and unsetting of bitmap pixels by means of signals from a "mouse" input device. In the screen view presented to the user, an editing area is set up. and an enlarged representation of the bitmap being edited is displayed within the editing area. Large squares represent each pixel in the bitmap. As these squares are set "on" or "off" during an editing session, an actual-size view of the symbol (displayed at the upper-right side of the editing area) is updated instantaneously to reflect each change made.

When an editing session for one symbol is complete, the user has the option to save his work by commanding BUILDFONT to put the new form of the symbol into the font management data structures, or he may discard the results of the editing session (in which case the previous form of the symbol remains stored in the data structures). The FONT EDIT module is called upon both when a new font is being created and when an existing font is beir revised (edited).

3. Editing a Font

BUILDFONT provides the means to change the data contained in an existing font file. This is the "EDIT" option of the Main Menu Level. Once BUILDFONT receives the name of the font file, the font is read into font memory

and all of its symbols are displayed as part of the "CHAR SELECT" module. At the same time, menu options are offered to perform the following editing tasks:

- ☞ EDIT -- the FONT EDIT module (described above) is called to edit the bitmaps of the selected symbol. If the selected symbol is undefined, the choice of this option is equivalent to adding a new character to the font.

- ☞ DELETE -- the selected character is simply removed from the font (i.e., its *raster array* and *font table* entries are erased), and its status in the CHAR SELECT display is changed to "undefined."

- ☞ PARMS -- with selection of this option, BUILDFONT calls upon a parameter editing sub-module which allows the user to adjust the "x-offset," "y-offset," and "skipwidth" of the selected symbol.

4. Displaying a Font

The "FONT DISPLAY" module is selected by the Main Menu Level option "DISPLAY." Within this environment, the user gets a view of all symbols in the font. The format of the display is similar to that of the CHAR SELECT module, however, no editing options are offered within the FONT DISPLAY module. FONT DISPLAY simply provides a way to get into and out of font files quickly. A user with a large number of font files stored on disk may find this feature useful in searching for one particular font or symbol.

5. The HELP Module

The BUILDFONT "HELP" feature can be accessed directly from most locations in other BUILDFONT modules. When HELP is needed, the user is presented with a "Help Menu" listing a number of HELP topics. On-line

explanations are printed out on the screen each time a topic is selected. The user has access to all of the HELP topics and explanations regardless of his current location within the BUILDFONT System.

D. EVALUATION OF THE BUILDFONT SYSTEM

Above, we stated that the BUILDFONT System can be evaluated for its functionality and user-friendliness by the number, convenience and efficiency of the operations it provides in accessing the font management data structures. To some extent this "functionality evaluation" is subjective: the determination of what constitutes "convenience" and "user-friendliness," on a conceptual level, is somewhat subject to individual preference. On the other hand, the number of operations incorporated into BUILDFONT and the system's run-time efficiency can be evaluated objectively, although we do so only descriptively in this study (using adjectives like "many," "few," "fast," "slow," etc.). In time, operating experience with the BUILDFONT System will produce both performance data, and hopefully, system improvements.

1. Font Editor Features and Operations

To put the BUILDFONT System's functionality into perspective, the user must imagine all the possible capabilities that can properly fall within the purview of a font editing utility and then judge the extent to which this particular font editor incorporates them. Based upon discussion presented earlier

in this study, from which we determined need for a font editor as a software development tool, the following listing of capabilities has been compiled:

MAJOR FUNCTIONS

- ☞ Create a font from scratch -- CREATE
- ☞ See what an existing font looks like -- DISPLAY
- ☞ Add new items to a font or delete existing items from a font -- ADD, DELETE
- ☞ Do "fine" editing on existing items in the font -- EDIT
- ☞ Change the order of items in a font, i.e., "scramble" the font items with respect to their correspondence characters -- SCRAMBLE
- ☞ Adjust parameters associated with a font table: maximum height, specific height, specific width, x-offset, y-offset, skipwidth -- PARM EDIT
- ☞ Merge two fonts, or pick items from one font and put them into another font -- MERGE, PICK

SUPPORT FUNCTIONS

- ☞ Manage support files:
  1. Locate or set up new font files -- SEARCH
  2. Protect work in progress -- STORE
- ☞ Provide editing modes:
  1. Pixel by pixel -- POINT
  2. Continuous -- CONSTANT
  3. Line rasterization -- LINE
  4. Whole components -- BLOCK
- ☞ Provide explanations and "Help" -- HELP

The reader is invited to augment this list with capabilities inadvertently omitted by the author.

Tables 5.1 and 5.2 present a comparison of the current BUILDFONT implementation features with the capabilities listed above. Of the

TABLE 5.1--SUMMARY FONT EDITOR MAJOR FUNCTIONS	
Capability	BUILDFONT Feature?
CREATE	Yes
DISPLAY	Yes
ADD	Yes
DELETE	Yes
EDIT	Yes
SCRAMBLE	No
PARM EDIT	Some
MERGE	No
PICK	No

nine major functions listed in Table 5.1, BUILDFONT incorporates the first five fairly comprehensively. Three of the major functions (SCRAMBLE, MERGE and PICK) have been omitted from the BUILDFONT System entirely. These were found by the author to be very challenging programming tasks which would have added unacceptable complexity to the BUILDFONT program, and their usefulness (if they had been incorporated) is in doubt. Nevertheless, future user experience may justify an effort to implement these functions into an improved version of the BUILDFONT System, and therefore they are listed here for completeness. The remaining capability, "PARM EDIT," is not fully incorporated into the current version of BUILDFONT. Initial setting of

maximum height, *average height*, *average width*, and interactive revision of *x-offset*, *y-offset*, and *skipwidth* are presently supported (as described earlier in this chapter). What BUILDFONT lacks in this area, however, is a capability to change the initial setting of *maximum height*. In addition, the bitmap height and width of an individual symbol may be increased up to only four pixels per call to the bitmap editor (FONT EDIT). For example, to thicken the top of the symbol "g" in figure 5.1 by nine pixels, the user would have to select "g" and go into the FONT EDIT environment three times (adding 4, 4, and 1 pixels, respectively). This is a bit clumsy, and so this area should receive the initial attention when program revisions are considered.

TABLE 5.2--SUMMARY
FONT EDITOR SUPPORT FUNCTIONS

Capability	BUILDFONT Feature?
SEARCH	Yes
STORE	Yes
POINT	Yes
CONSTANT	Yes
LINE	No
BLOCK	No
HELP	Yes

Of the support functions listed in Table 5.2, SEARCH, STORE, and HELP are considered adequate, as are the POINT and CONSTANT bitmap editing modes. The LINE and BLOCK modes, presently not implemented, would add considerable convenience to the FONT EDIT module. The idea behind the BLOCK editing mode is to have a file of pre-existing symbol components from

which items can be selected and "dropped" into a bitmap in the BUILDFONT editing area to form a new. more complex symbol (built up from components). This editing mode is particularly suited for rapidly creating a large symbol set containing complex characters. such as the character set for written Chinese. Thus. attention to improving the BUILDFONT System's editing modes is another priority item to be included in plans to upgrade the system.

2. Run-time Considerations

The BUILDFONT System's run-time performance has generally been adequate. However, in the present implementation, the program requests large amounts of RAM memory, and editing large bitmaps (30x30 pixels and larger) tends to affect editing speeds quite adversely. For example, editing a 40x40 bitmap causes the creation of 1600 graphics objects and a like number of editing "tags" [cf. Ref. 18: chap. 8]. All 1600 objects must be displayed instantaneously after each change made in the editing area. The complexity of this procedure with respect to time and space is of the order of $O(n^2)$, and the dropoff in performance with large bitmaps is noticeable to the user.

The main purpose of the present implementation of the BUILDFONT System is to demonstrate the capabilities that have been incorporated into the program, and various excessive run-time overhead costs have yet to be eliminated. One of the major overhead costs is the present menu system, developed in the Naval Postgraduate School's Graphics and Video

Laboratory.[21] Since memory costs for BUILDFONT are already high, the system can benefit from the design and implementation of a less general, tailor-made menu system.

Overall, the BUILDFONT System addresses a need which has gone unsatisfied heretofore on the IRIS-2400 Graphics Workstation, and positive user experience is expected, at least in the near term.

E. IMPLEMENTATION DETAILS

The BUILDFONT System implementation is comprised of 29 program files, 6 support files, and a total of over 10,000 lines of "C" code. Liberal commentary is interspersed throughout the source code, and format and naming conventions of the code generally adhere to the style used in example programs from graphics courses taught at the Naval Postgraduate School. These characteristics were intentionally incorporated into the program to enhance familiarity and readability, and to encourage borrowing from and follow-on improvements to the BUILDFONT System. References 19 and 20 were regularly consulted during preparation of the program code. The source code and documentation for the current version of the BUILDFONT System is retained in the Naval Postgraduate School Graphics and Video Laboratory (with public domain access).

[21] The general menu file is called *menu.files.c*, written by M. Gaddis in 1984.

VI. CONCLUSIONS AND RECOMMENDATIONS

This study has presented a discussion of the characteristics of symbol manipulation systems. These systems, whether sophisticated or rudimentary, are a fundamental component of every computing installation. We have touched upon application areas where the present generation of symbol manipulation systems suffers from inherent limitations, and we have suggested a need to remove these limitations by increasing the generality of present systems. We have determined that symbol manipulation systems seem to be limited more by their design approach than by any underlying technological or hardware constraints. In this observation there is hope that more general symbol manipulation systems can be brought about by demonstrating more creativity and sophistication in the development of symbol manipulation-oriented applications, rather than by waiting for technological breakthroughs alone.

Concurrent with background research into symbol manipulation systems, the BUILDFONT Font Creation and Editing System has been developed as a tool to assist future efforts in this area. The purpose of the BUILDFONT System is to support IRIS System applications requiring customized symbol fonts by easing the task of creating and editing these fonts.

This study has brought together source materials from several important areas: linguistics, word processing, and computer graphics, and the present report has attempted to emphasize a descriptive approach to the subject matter, without detailed investigation in any one area. It is felt that such an approach will be of greater benefit to future researchers in this area than an isolated study concerned with narrower subject matter. Obviously, much work in the area of improving symbol management systems remains to be done. Some immediate follow-on topics come to mind:

In computer graphics, the following architectural/design issues need to be addressed:

- ☞ Augmenting font operations in "leading-edge" graphics systems, to bring font manipulation techniques up to the level of overall system sophistication.

- ☞ Increasing font memory capacity to accommodate more symbols.

In the area of software development, the following tasks remain to be addressed:

- ☞ Implementing the Ideal Word Processor concept discussed in Chapters 3 and 4.

- ☞ Creating additional software support tools like the BUILDFONT System to facilitate font management research.

Improved versions of the BUILDFONT System will result from the following:

- ☞ Development of more convenient operations to revise the *font table* parameters (bitmap *height* and *width*, and font *maximum height*.)

- ☞ Implementation of the LINE and BLOCK editing modes described in Chapter 5.

- ☞ Improving run-time performance by eliminating inefficiencies present in the current (prototype) version.

APPENDIX A – GLOSSARY OF TERMS

The topics presented in this study are drawn from several fields: computer graphics, word processing, linguistics, and Asian area studies. This glossary of terms and abbreviations has been compiled to assist the reader who may be unfamiliar with the technical vocabulary of some of these subject areas. In cases where an explanation or definition is quoted directly from reference materials, the source is cited. Where foreign words are defined, the English translation appears in the explanation first (in quotation marks), followed by the language from which the term is taken (in parentheses), followed by the definition. Italicized words appearing in any of the definitions and explanations have their own entries in this glossary.

agglutinative languages

Languages which *synthesize* or bind together strings of *morphemes* to form grammatical structures. These structures are the "units" of the language, although in terms of complexity, they are larger than *word-units* (but they are less than the size of a full sentence). Japanese is an agglutinative language. The difference between an agglutinative language and an "inflectional language" (e.g., Latin) is that the morphemes retain their integrity of meaning in an agglutinative language, and they do not fuse into variant forms even though they are bound together. (see also *typological classification*)

analytic languages

Languages of the "isolating" type (see *typological classification*), which have the characteristic that all words tend to be simple roots, and the grammatical category of a word is determined primarily by its position in a phrase or sentence structure. Classical Chinese is a nearly perfect example of this type, while modern Chinese dialects and English fall within this category even though they permit many forms *synthesized* from more than one root *morpheme*.

ASCII

"American Standard Code for Information Interchange." One of two standard symbol coding conventions commonly used in computing and communication systems of the United States and Western Europe (the other is *EBCDIC*). It provides codes for up to 128 symbols and control signals. Reference 20 (Appendix 11) includes a table of correspondences between the symbols/control signals and the numerical codes.

bitmap

A two-dimensional array of computer data words which can be mapped electronically onto the display.

Bopomofo

A popular abbreviation for "*Chinese National Phonetic System*" (*bo, po, mo,* and *fo* are sounds corresponding to the first four symbols in the set of phonetic symbols).

bunsetsu

(An untranslatable term from Japanese linguistics.) *Bunsetsu* are the independent units from which sentences are formed in Japanese--roughly analogous to "phrases" in English. (see also *agglutinative languages*)

CCCII

"Chinese Character Code for Information Interchange." One of three standard symbol coding conventions used in Chinese (*ROC*) word processing and communication applications. (see also *Chinese Standard Cable Code, GB 2312-80*)

Chinese National Phonetic System

A set of 37 phonetic symbols used to transcribe spoken Chinese (Standard Mandarin dialect) into written form. Each symbol corresponds to a combination of one or more *phonemes* (most of the symbols represent a unit of sound which is shorter than a syllable; i.e., syllables are expressed with one to three symbols). Chinese call the National Phonetic System *zhuyin-fuhao, zhuyin-zimu,* or *bopomofo.*

Chinese Standard Cable Code

One of three standard symbol coding conventions used in Chinese (*PRC*) word processing and communication applications, particularly telegraphy. The character set supported by this convention contains 8085 symbols. (See also *CCCII, GB 2312-80*)

EBCDIC

"Extended Binary-Coded Decimal Interchange Code." One of two standard symbol coding conventions used in computing and communication systems of the United States and Western Europe (see *ASCII*). It provides codes for up

to 256 symbols and control signals (although only about 194 codes are actually used). EBCDIC is commonly used with equipment manufactured by IBM. Reference 21 (Appendix B) provides a table of correspondences between the symbols/control signals and the numerical codes.

firmware

Non-hardware features of a computer installation which are permanently stored and protected (in ROM memory). The IRIS System default *font* is an example of firmware.

font

A set of characters in a particular style. [Ref. 18: Glossary, p. 4]

font memory

A reserved and protected hardware computer memory set aside to contain character data which can be mapped directly into the *frame buffer* of a raster refresh graphics system.

font table

A program data structure (often system-defined) used to store parameters describing the symbols of a *font*.

frame buffer

A specialized memory containing all pixels and pixel attributes in a raster refresh graphics system. This array of pixels is mapped directly to corresponding points on the display surface.

GB 2312-80

Designation for "Information Exchange for Chinese Character Codes (Basic Volume)." One of three standard symbol coding conventions used in Chinese (*PRC*) word processing and communication applications. The character set supported by this convention contains 6763 symbols. (See also *CCCII, Chinese Standard Cable Code*)

hangul

The phonetic transcription system by which the Korean language is written.

Hanyu-pinyin

"Chinese spelling" (Chinese). A system of *romanization* for Standard (Mandarin) Chinese. This is the official system of the Chinese government (*PRC*). All Chinese words appearing in this study have been transcribed according to this system.

hanzi

"Chinese characters" (Chinese). (see also *kanji*)

hiragana

A *syllabary* containing 53 symbols used for transcribing spoken Japanese into written language. Hiragana have a cursive (handwritten) appearance.

homophone

A syllable or word which is pronounced identically to another syllable or word in the same language (even though the meanings are different). For example. in English. "ball" (the round thing) and "ball" (where Cinderella lost her slipper) are homophones.

homophone resolution problem

The problem of deciding which word or syllable is meant when a speaker or writer uses a homophone.

input problem

The problem of converting input symbols into a consistent internal representation in a computer system.

Designation for "Code of the Japanese Graphic Character Set for Information Interchange." The standard symbol coding convention used in Japanese communication and word processing applications.

kana

Sets of phonetic symbols used to transcribe spoken Japanese language into written form. These symbol sets are *syllabaries.* (see also *hiragana, katakana*)

kanji

"Chinese characters" (Japanese). (see also *hanzi*)

katakana

A *syllabary* containing 53 symbols used for transcribing spoken Japanese, particularly emphasized words or foreign loan words and names (except those borrowed from Chinese). Katakana have an angular (printed) appearance.

morpheme

The smallest subdivision of language which carries meaning. A morpheme is not necessarily an independent unit (except in "isolating" languages). For example, in the word *flyer*, the morpheme *fly* is independent (i.e., a "word"), but the morpheme *-er* is not (it is a "bound form"). [Ref. 1: p.51]

output problem

The problem of converting the internal representation of information within a computer into symbols that are meaningful in the external (user's) environment.

PRC

"Peoples Republic of China." Formal name for the political entity controlling Chinese territory on the Asian mainland.

phoneme

An element from the set of sounds ("phonemic system") utilized by a spoken language. More formally, a *phoneme* is "one of an exhaustive list of systematized classes of phonetically related sounds in a language, such that every form in the language can be given as a (usually serially ordered) set of one or more of these classes [Ref. 1: p. 37]

pixel

"Picture element," or individual data item from which a graphics object is formed in a raster refresh graphics system. Pixels are in one-to-one correspondence with points on the display screen.

Qwerty keyboard

The arrangement of keys found on a standard Roman-character typewriter. This arrangement was developed in the early 1900's and takes its name from the first six keys on the left side of the uppermost row of alphabetic keys.

ROC

"Republic of China." Formal name for the political entity controlling the island of Taiwan.

romaji

"Roman characters" (Japanese). The Japanese term for *romanization.* All Japanese words appearing in this study have been transcribed by the Hepburn System of *romaji.* (see also *romanization*)

romanization

A method of phonetic transcription of oriental languages whereby the sounds of the spoken language are represented by combinations of letters from the Roman alphabet. Some important and/or historically significant systems of romanization are: *Hanyu-pinyin*, the Wade-Giles System, the Yale System, and the National Romanization System (also called the Chao System) -- for Chinese; the Hepburn System -- for Japanese.

Sino-Japanese language

The set of Chinese words which have been borrowed and assimilated into Japanese. The Japanese pronunciation of these words (called the "On"-reading) requires a special phonemic system which augments the natural phonemics of Japanese.

syllabary

A set of phonetic transcription symbols with each element of the set representing the sound of a complete syllable in the language being transcribed.

synthetic languages

Languages of the "inflectional," *agglutinative*, or "polysynthetic" type (see *typological languages*). These languages "synthesize" the grammatical forms and independent units of the language by combining root *morphemes*.

TCCM

The "Three-Corner Coding Method." A non-phonetic numerical coding method used to convert Chinese characters into internal codes in a word processing system. The codes are based on selecting three "corners" (significant components) contained in an individual Chinese character from a set of 300 fundamental symbols. Each of the fundamental symbols has a two-digit code, so a complete code for one Chinese character is six digits long.

typological classification

A method of comparing and classifying languages according to their types of structure (regardless of whether or not they are genetically related). The best known system utilizing this method classifies languages into the following types: (a) isolating (or analytic), (b) inflectional, (c) agglutinative, and (d) polysynthetic. [Ref. 1, pp. 87-89]

Wabun keyboard

A tablet comprised of several thousand keys used with Chinese and Japanese language typewriters. The *Wabun* keyboard was invented in Japan in 1913.

Word processing problem

The *input problem*, the *output problem*, and the problem of how to make a computer system manipulate signal codes internally to perform word processing.

word-unit languages

Languages in which the smallest form that can be used independently (i.e., not necessarily in combination with other forms) is the "word." For example, Classical Chinese is a *morpheme*-unit language: English is a word-unit language; Japanese is a *bunsetsu*-unit language.

zhuyin-fuhao

"Phonetic symbols" (Chinese). (see *Chinese National Phonetic System*)

zhuyin-zimu

"Phonetic symbols" (Chinese). (see *Chinese National Phonetic System*)

APPENDIX B – IRIS2400 SYSTEM CHARACTERISTICS

A. SYSTEM DESCRIPTION

The Graphics and Video Laboratory of the Naval Postgraduate School Computer Science Department is equipped with two IRIS-2400 series Graphics Workstations, manufactured by Silicon Graphics, Inc. One of these workstations is an upgraded IRIS Turbo 2400. The BUILDFONT Font Creation and Editing System software is hosted on both of these IRIS-2400 configurations.

The IRIS System incorporates custom-built VLSI circuits into its design, providing special-purpose hardware processing elements to perform many computer graphics functions which are done by less efficient software in conventional workstations. As a result of its innovative architecture, the IRIS System offers high processing speeds and increased performance reliability in the execution of computer graphics applications programs. The IRIS System combines real-time color graphics with UNIX operating system utilities and ethernet network communication. In addition, the System has a high resolution color monitor which provides extremely sharp, well-defined displays capable of supporting the requirements of very demanding graphics applications.

The IRIS System includes a Graphics Library of utility programs and subroutines, creating a user-friendly interface between the programmer and the sophisticated hardware features, such that graphics objects can be handled as

geometrical abstractions (points. lines. polygons. etc.). rather than formless collections of pixels. The System also manages multiple coordinate systems. allowing users to define objects within the "world space" of their applications. IRIS Graphics Library subroutines used in the BUILDFONT System can be grouped into the following categories (a complete explanation of command usage is contained in Reference 18):

- *Global state* commands initialize the hardware and control global state variables.
- *Primitive drawing* commands draw points, lines, polygons, circles. arcs, and text strings into graphics objects that can be drawn on the screen.
- *Coordinate transformation* commands perform manipulations on coordinate systems. including mapping user-defined coordinate systems to screen coordinate systems.
- *Input/output* commands initialize and read input/output devices.
- *Object creation and editing* commands provide the means to create hierarchical structures of graphics commands.
- "Picking and selecting" commands identify the commands that draw to a specified area of the screen.

B. IRIS SYSTEM SPECIFICATIONS AND FEATURES

The IRIS-2400 series Graphics Workstations in the Naval Postgraduate School Computer Science Department Graphics and Video Laboratory are configured as follows:

1. IRIS-1

- IRIS-2400 Graphics Workstation
- 32-bit Motorola 68010 Processor
- 4MB CPU Memory
- 1024 x 768 x 8 bit display memory
- Floating Point Accelerator
- 144MB Disk Storage
- Cartridge Tape Unit
- Geometry Pipeline with Geometry Engines and Geometry Accelerators
- 30-Hz Interlaced Display
- Hardware Smooth Shading
- UNIX System V
- IRIS Graphics Library
- Ethernet to VAX installations

2. IRIS-2

- IRIS Turbo 2400 Graphics Workstation
- 32-bit Motorola 68020 Processor
- 2MB CPU Memory
- 1024 x 768 x 32 bit display memory
- Floating Point Accelerator
- 144MB Disk Storage
- Cartridge Tape Unit
- Geometry Pipeline with Geometry Engines and Geometry Accelerators
- 60-Hz Non-Interlaced Display
- Hardware Smooth Shading
- UNIX System V
- IRIS Graphics Library
- Ethernet to VAX installations
- 16-bit Z-buffer for Hidden Surface Elimination
- Digitizer Tablet

LIST OF REFERENCES

1. Chao, Y. R., *Language and Symbollic Systems*, Cambridge University Press, 1968.

2. Huang, J. K., "The Input and Output of Chinese and Japanese Characters," *Computer*, v. 18, January 1985.

3. Becker, J. D., "Typing Chinese, Japanese, and Korean," *Computer*, v. 18, January 1985.

4. Kennedy, G. A., *ZH Guide -- An Introduction to Sinology*, Far Eastern Publications, Yale University, 1953.

5. Matsuda, R., "Processing Information in Japanese," *Computer*, v. 18, January 1985.

6. *A Guide to Reading and Writing Japanese*, 2nd ed., edited by F. Sakade, and others, Charles E. Tuttle Company, 1961.

7. Kim, C. H. and Ko, S. W., *Implementation of Korean and Chinese Characters through Computer*, M. S. Thesis, Naval Postgraduate School, Monterey, California, September 1984.

8. Lee, J. H., *A System for Korean Character Usage on a Graphics Laser Printer*, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1986.

9. Bond, N. A., Jr., "Yamada's Remarkable Keyboard," *ONRFE Scientific Bulletin*, v. 10(1), January - March 1985.

10. Morita, M., "Japanese Text Input System," *Computer*, v. 18, May 1985.

11. Sheng, J., "A Pinyin Keyboard for Inputting Chinese Characters," *Computer*, v. 18, January 1985.

12. Makino, H., "Beta: An Automatic Kana-Kanji Translation System." *Computer*, v. 18, January 1985.

13. Tien, H. C., "The Pinxxiee Chinese Word Processor." *Computer*, v. 18, January 1985.

14. Yajima, S., Goodsell, J. L., Ichida, T. and Hiraishi, H., "Data Compression of Kanji Character Patterns Digitized on the Hexagonal Mesh," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. PAMI-3(2), March 1981, pp. 222-229.

15. Naval Postgraduate School Report NPS52-85-012, *Workstation Graphics Capabilities for the 1990's and Beyond*, by M. J. Zyda, September 1985.

16. Rogers, D. F., *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, 1985.

17. Clark, J. H. and Davis, T., "Workstation Unites Real-time Graphics with Unix, Ethernet," *Electronics*, October 20, 1983.

18. *Iris User's Guide*, version 2.1, Silicon Graphics, Inc., 1985.

19. Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, Inc., 1978.

20. Bourne, S. R., *The UNIX System*, Addison-Wesley Publishing Company, 1983.

21. Flores, I., *Word Processing Handbook*, Van Nostrand Reinhold Company, 1983.

BIBLIOGRAPHY

Bond. N. A., Jr.. "Automatic Recognition of Handprinted Chinese-Japanese Kanji: The Last Frontier of Chinese Character Recognition?," *ONRFE Scientific Bulletin*, v. 9(3), July - September 1984.

Cui. W., "Evaluation of Chinese Character Keyboards," *Computer*, v. 18, January 1985.

Foley. J. D. and Van Dam. A.. *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company. 1982.

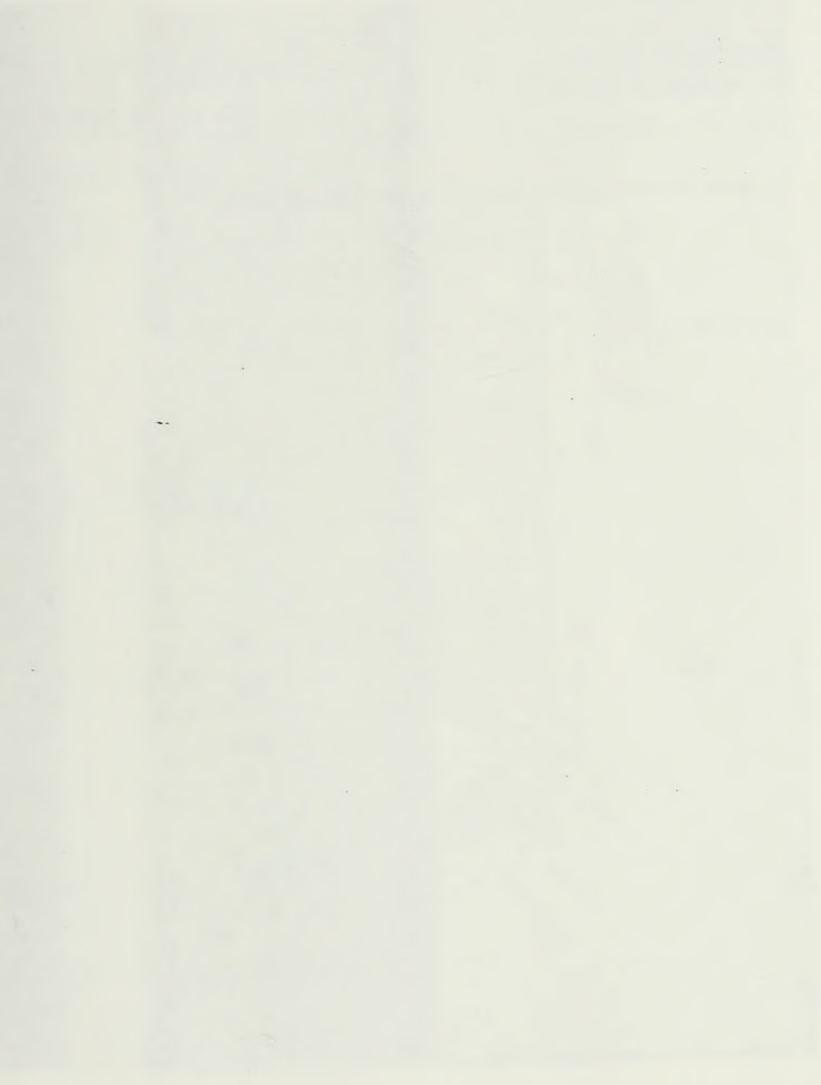
Friedman. N. K., "Japanese Word Processing: Interfacing with the Inscrutable," *Abacus*, v. 3(2). Winter 1986.

Mathew's Chinese-English Dictionary, Revised American Ed., Harvard University Press, 1971.

Nelson, A. N., *Japanese-English Character Dictionary*, 2nd ed., Charles E. Tuttle Company. 1966.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Superintendent Attn: Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Computer Technology Curricular Officer, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Dr. Michael J. Zyda, Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
6.	Dr. C. Thomas Wu, Code 52Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	LT James C. Artero, USN Naval Ship Weapon Systems Engineering Station Port Hueneme, California 93043-5007	5



218671

Thesis
A755
c.1

Artero
Non-Roman font generation via interactive computer graphics.