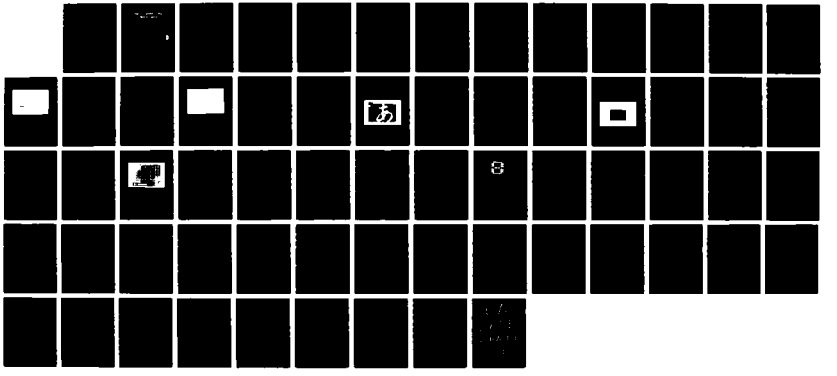
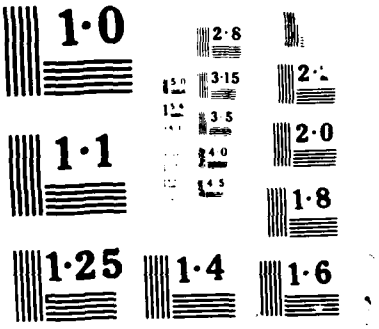
AD-A191 007 EDITFONT AN INTERACTIVE FONT EDITING SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA H J OLIVERA DEC 87 1/1

UNCLASSIFIED F/G 12/5 NL




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·8
3·15
3·5
4·0
4·5

AD-A191 007

DTIC FILE COPY

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DTIC
ELECTE
MAR 2 4 1988
S D
E

EDITFONT
AN INTERACTIVE FONT EDITING SYSTEM

by

Hector Mariscal O.

December 1987

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited.

88 3 22 042

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) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 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)
EDITFONT AN INTERACTIVE FONT EDITING SYSTEM (u)

12 PERSONAL AUTHOR(S)
Mariscal, Hector O.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 December	62

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Font editor; Icons; Font generation; Font extraction

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The major goal of this work has been the development and implementation of an interactive bit mapped font editor that enables the graphics programmer to create different fonts and icons for the use in application programs. The font editor, called editfont, has been implemented on the Silicon Graphics, Inc. IRIS workstation. Editfont consists of approximately 3500 lines of code, including the program documentation. Fonts created by editfont can be retrieved from disk and high level routines implemented with the IRIS graphics library. One feature of the system is the capacity for font extraction from a picture. The steps for font generation via font extraction are explained in detail. File format, data structures and routines used by the system are also described. Software and hardware limitations of the system are outlined, as well as possible future extensions.

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

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

U.S. Government Printing Office 1986—606 24.

Approved for public release; distribution is unlimited.

Editfont
An interactive font editing system

by

Hector J. Mariscal Olivera
Teniente Segundo, Peruvian Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

December 1987

Author: _____
Hector J. Mariscal

Approved by: _____
Michael J. Zyda, Thesis Advisor

_____
C. Thomas Wu, Second Reader

_____
Vincent Y. Lum, Chairman
Department of Computer Science

_____
James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

The major goal of this work has been the development and implementation of an interactive bit mapped font editor that enables the graphics programmer to create different fonts and icons for use in application programs. The font editor, called **editfont,** has been implemented on the Silicon Graphics, Inc. IRIS workstation. **Editfont** consists of approximately 3500 lines of code, including the program documentation. Fonts created by **editfont** can be retrieved from disk and high level routines implemented with the IRIS graphics library. One feature of the system is the capacity for font extraction from a picture. The steps for font generation via font extraction are explained in detail. File formats, data structures and routines used by the system are also described. Software and hardware limitations of the system are outlined, as well as possible future extensions.

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION 7
A. OVERVIEW 7
B. BACKGROUND 7
C. ORGANIZATION 9
II. EDITFONT : SYSTEM OVERVIEW 11
A. STARTING EDITFONT 11
B. MAIN MENU 11
1. Selecting a file from the directory 11
2. Options on the main menu 13
a. COPY 13
b. NEW 13
c. DELETE 13
d. EDIT 14
e. SET PICTURE 14
f. EXIT 14
C. CHARACTER SELECTION MODE 14
1. ASCII correspondence and example area 14
2. Selecting a character 17
3. Options on the character selection mode 17
a. OPEN 17
b. PICTURE 17
c. SAVE 17
d. ABORT 17
D. CHARACTER EDIT MODE 18
1. Pen options 19
a. Xor pen type 19
b. Xand pen type 19
2. Line options 19
a. Drawing lines 19
b. Erasing lines 20
3. Scaling 20
4. Changing the bitmap size 20
5. Moving the character around 20
6. Copying a character 21
7. Undoing the last command 21
8. Erasing the bitmap 21

9. Changing the character parameters 21
10. Saving the character bitmap 23
E. EDITFONT AS AN ICON EDITOR 23
F. SYSTEM WARNING SIGNALS 24
III. FONT GENERATION FROM AN IMAGE 25
A. TAKING PICTURES FOR EDITFONT 25
B. USING THE PICTURES FOR FONT EXTRACTION 26
C. FONT EXTRACTION MODE 27
1. Selecting the ASCII correspondence 27
2. Changing the brightness of the image 29
3. Changing the size of the lens 29
4. Extracting a character 29
5. Moving the image 30
6. Exiting font extraction mode 30
D. IMAGE FILE FORMAT 30
IV. FONT UTILIZATION 32
A. SYSTEM LIBRARY ROUTINES 32
1. defrasterfont 32
2. font 33
3. getfont 34
4. getheight 35
B. HIGH LEVEL ROUTINES 35
1. fontdef 35
2. delfont 35
C. AN EXAMPLE PROGRAM 36
D. FONT FILE FORMAT 37
V. SYSTEM CONSIDERATIONS AND CONCLUSIONS 41
A. SOFTWARE LIMITATIONS 41
B. HARDWARE LIMITATIONS 42
C. CONCLUSIONS AND RECOMMENDATIONS 42
APPENDIX A - KANJI FONT FILE CREATED BY EDITFONT 44
APPENDIX B - FONT DATA STRUCTURE 54
APPENDIX C - HIGH LEVEL ROUTINES 55
LIST OF REFERENCES 60
INITIAL DISTRIBUTION LIST 61

LIST OF FIGURES

1.1 Font handling routines 10
2.1 Main menu display 12
2.2 Character selection mode 15
2.3 Non-printable characters display 16
2.4 Character edit mode 18
2.5 Character parameter mode 22
3.1 Font extraction mode 28
4.1 Display of a character 34
4.2 Example program 38
4.3 Example font file 39

I. INTRODUCTION

A. OVERVIEW

The Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School is equipped with several high performance graphics workstations manufactured by Silicon Graphics, Inc. of Mountain View, California. These workstations are based on the Motorola 68020 processor. The workstations have a graphics library. One of the major deficiencies of the IRIS workstation is its relatively low-level support functions for defining fonts. Additionally, the system comes with only a single 9 by 9 bitmapped font. It is the purpose of this study to improve the support on the IRIS for font and icon construction. For that goal, we have implemented in the C programming language a font editor, **editfont,** and a set of font support software. **Editfont** is a full featured font editor with capabilities for font definition through a **paint-like** interface and for font extraction from digitized images. The font support software is constructed on the low-level font definition routines available in the IRIS graphics library. The support software reads into IRIS font memory from disk fonts defined by **editfont.**

B. BACKGROUND

The IRIS workstation has the capacity for users to define different fonts for application programs. It provides a set of utility commands implemented as routines callable from the "C" language. These routines are :

defrasterfont : define a raster font.

font : selects the desire font.

getfont : returns the number designating the font currently in use.

getheight : returns the maximum height value of the font

strwidth : returns the width (in pixels) of the text string.

The **defrasterfont** function is the one that defines a raster font. This function loads the font information from main memory into the special IRIS font memory. This routine has six input parameters. This information includes sizes of each character, the bitmap information and the relative position of the bitmap with respect to the current character position pointer. Besides this routine, no other support is provided for font definition. No documented file format or font editor is provided. There is a need for a tool that enables the user to create complex fonts and store them on disk as well as high level routines that can call the defined fonts.

The **editfont** font editor is a system that runs on the Silicon Graphics, Inc. IRIS workstation. **Editfont** has been implemented applying the concept of user friendly interfaces. It uses the mouse as its primary input and is a window driven system that detects any user input errors and warns the user by displaying messages or by beeping the alarm bell. Chapter II explains the use of **editfont** to create fonts.

Font generation via font extraction from images is a feature of **editfont.** This feature enables the user to extract different characters from a picture taken with a digitizer camera. Chapter III explains the steps necessary for font extraction.

High level routines are needed to reduce the complexity of defining fonts from that provided in the IRIS package. There are two routines available for the programmer to use in his application program :

fontdef : loads a font file from disk to font memory.

delfont : erases the font definition from font memory.

A detailed description of these routines is provided in Chapter IV. Figure 1.1 shows how the font information is handled by the different parts of the system. **Defrasterfont** loads the font information from main memory to font memory. **Editfont** creates fonts in disk files and the **fontdef** routine loads the information from disk directly to font memory. The dashed lines indicate that internally this routine has to read the information first to main memory and then move it to font memory. This data movement is transparent to the user.

C. ORGANIZATION

The above sections have provided an overview of the support tools for font definition available in the IRIS workstations, and how can this support be improved. Chapter II provides information on how to use the **editfont** system for font construction. Chapter III covers the steps necessary for font generation via font extraction. Chapter III also includes information about the image format used by **editfont,** and how to create compatible image files. Chapter IV covers font utilization using the system functions and the high level routines. Chapter IV also explains the different parameters needed to define a font. Chapter IV also shows the font file format used by the font editor. Chapter V covers the system's limitations and future recommendations.

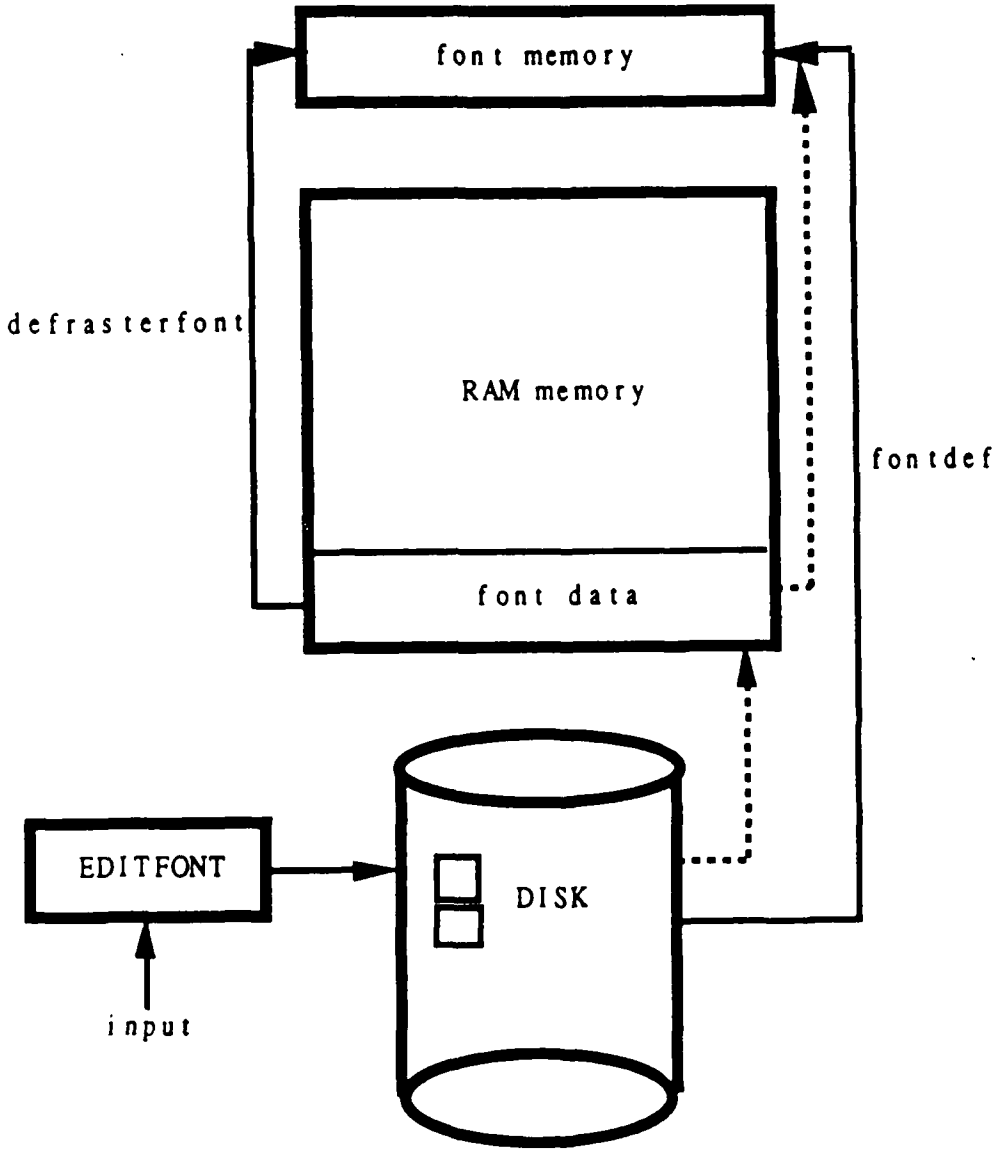


Figure 1.1 Font handling routines

II. EDITFONT : SYSTEM OVERVIEW

Editfont is a simple, easy to learn, easy to use font editor implemented on the IRIS workstation. The system is menu driven and commands are entered using the mouse device. Users errors are checked by the system and appropriate warning messages as well as sound signals are issued. **Editfont** consists of four main windows or modes which the user can enter at any time. These modes are: the main menu, character selection mode, character edit mode and font extraction mode.

A. STARTING EDITFONT

To start the font editor, one types "editfont" on the IRIS. The main menu then appears on the screen. The user must be in the directory in which the font files reside.

B. MAIN MENU

The first screen shown by the system is the main menu. The MIDDLEMOUSE is used to select options from this menu. Figure 2.1 shows how this window is displayed on screen. The main menu contains six options, mostly for handling font files, and a directory window that displays the file names stored in the current directory.

1. Selecting a file from the directory

At the main menu, the system displays the current directory in a small window. At most, 15 file names can be seen in this window. If the user needs to see more file names, the directory can be scrolled by clicking inside the upward or downward arrows.

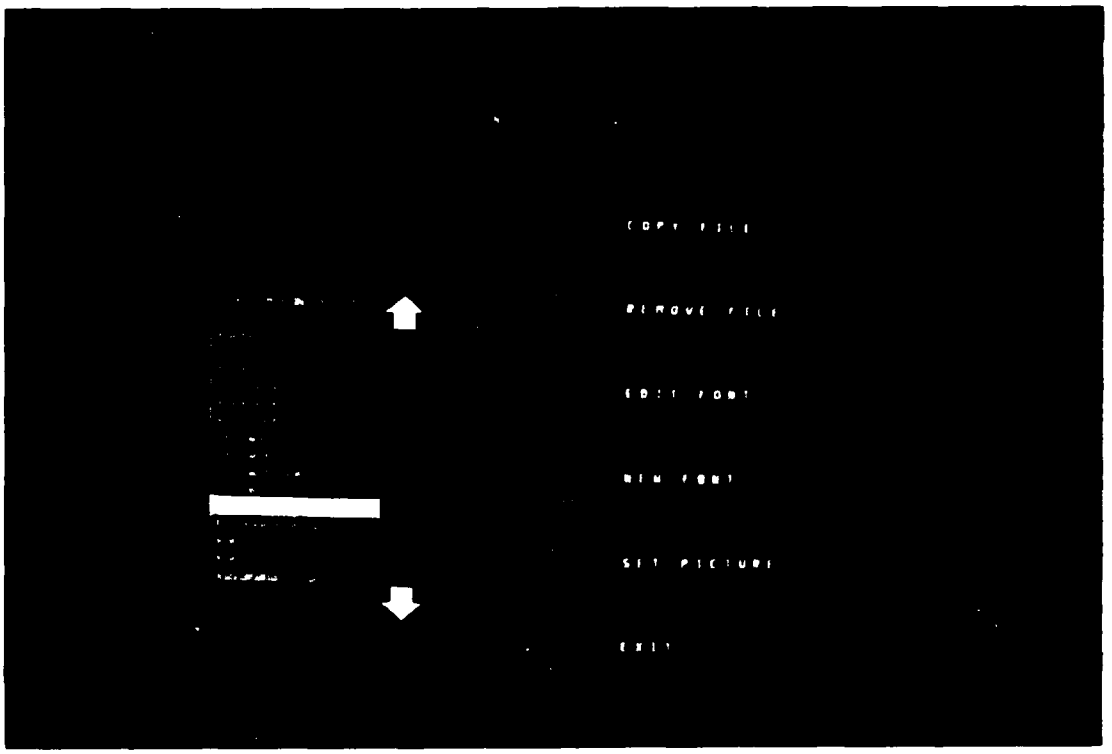


Figure 2.1 Main menu display

To select a font file or a picture file, the cursor must be moved so it points to the desired file name. Then MIDDLEMOUSE must be pressed. The selected file is then highlighted. **Editfont** takes this file name for later operations.

2. Options on the main menu

a. COPY

This option allows the user to copy a font file into a file of a different name. To copy a file, the user has to select one from the directory window and then select the copy option. The system then prompts the user for the name of the new file. The user can enter any name using the keyboard. The system does not accept special characters as part of the name, it ignores such characters. If the user hits the carriage return key without typing a name, the system returns to the main menu without creating a new file. The copy option is not restricted to act only for font files, any file selected by the user can be copied.

b. NEW

This option allows creation of a new font file. When activated, a prompt appears asking for the name of the new file to be created. After the file name has been entered, the system is placed into character selection mode. This mode is described below.

c. DELETE

This option is used to erase any undesired font file from disk. To use this option, a file must be selected and then the MIDDLEMOUSE clicked on DELETE. At this point, a warning window appears, showing the name of the file that is going to be erased. The user has the option of clicking in CONTINUE, to erase the file, or clicking in ABORT to cancel the delete option. This command deletes any selected file, including non-font files.

d. EDIT

This option allows the modification of an existing font file. By selecting a font file and clicking on EDIT, the system is put into character selection mode. If the selected file is not a font file, the system warns the user by ringing the keyboard bell.

e. SET PICTURE

This option allows the user to select a picture file name for character extraction. The system does not check if the file is a valid picture file until the PICTURE option is selected on the character selection menu. Information about character extraction from a picture can be found in Chapter III.

f. EXIT

This option terminates the execution of **editfont.**

C. CHARACTER SELECTION MODE

Character selection mode is entered when the user selects NEW or EDIT on the main menu. This mode is used to select the characters that the user wants to edit or create. This mode displays the font file name that is currently being used, the ASCII correspondence characters, and an example of how the font characters currently look. Figure 2.2 shows how this window is displayed on screen.

1. ASCII correspondence and example area

The ASCII correspondence characters that are printable characters are displayed. Non-printable characters are displayed and marked specially. Figure 2.3 shows how the non-printable characters are displayed in character selection mode.

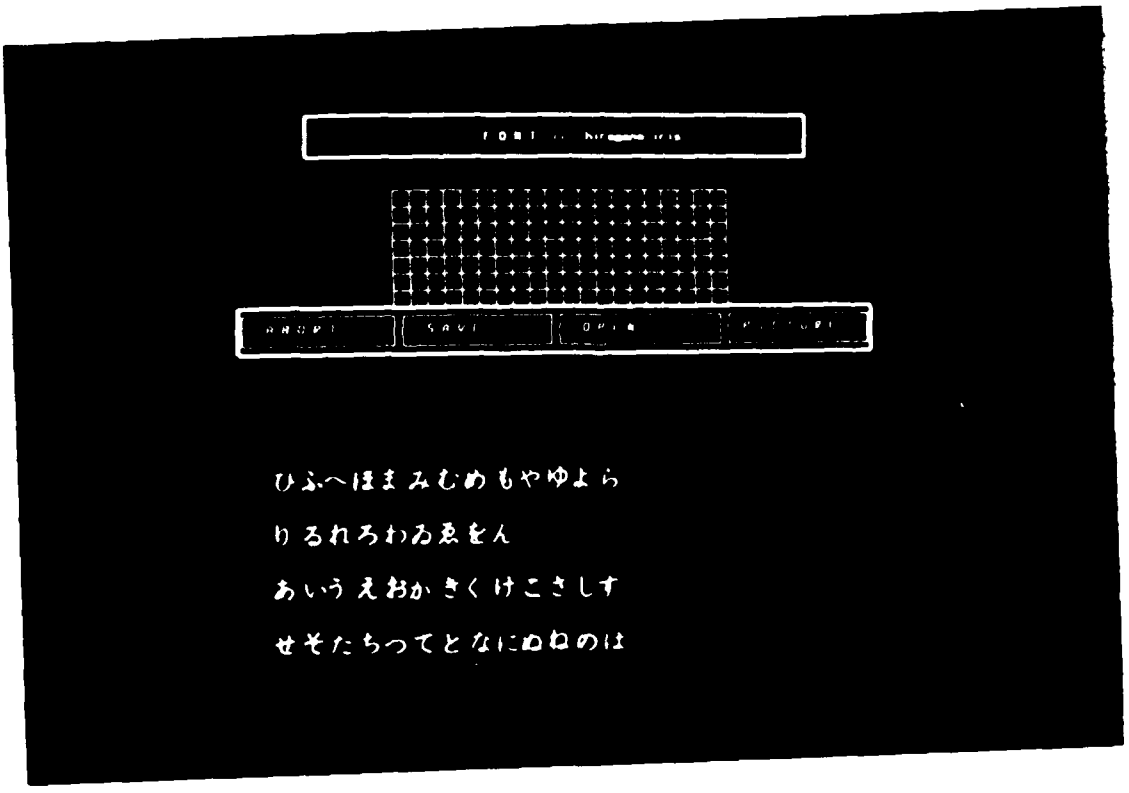


Figure 2.2 Character selection mode

000 = nil	017 = ^Q
001 = ^A	018 = ^R
002 = ^B	019 = ^S
003 = ^C	020 = ^T
004 = ^D	021 = ^U
005 = ^E	022 = ^V
006 = ^F	023 = ^W
007 = ^G	024 = ^X
008 = ^H	025 = ^Y
009 = ^I	026 = ^Z
010 = ^J	027 = ^[
011 = ^K	028 = ^/
012 = ^L	029 = ^]
013 = ^M	030 = ^^
014 = ^N	031 = ^_
015 = ^O	032 = blank
016 = ^P	127 = ^?

Figure 2.3 Non-printable characters display

The example area displays font characters already created. When the user enters character selection mode, all the defined alphabetical characters are displayed in the example area. To see the rest of the characters (non-printable and special characters), the user must click inside the rectangle containing the word "example".

2. Selecting a character

To select a character in this mode, the user has to click inside the square in which the ASCII correspondence character is located. The selected character is then highlighted. Another way of selecting a character is by clicking on the characters displayed as examples. This mode is useful for when the font on which we are working is non-Roman.

3. Options on the character selection mode display

a. OPEN

The OPEN option displays the bitmap of the selected character as filled polygons or "fat bits". When this option is selected, the system enters edit mode. Edit mode is explained below.

b. PICTURE

The PICTURE option transfers control to character extraction mode. To enter this mode, the user should previously have selected a picture file from the main menu. For information about character extraction techniques, see Chapter III.

c. SAVE

The SAVE option stores all the changes that have been made to the current font. This option returns the user to the main menu window.

d. ABORT

The ABORT option does not save the changes to the current font file. This option returns the user to the main menu window.

D. CHARACTER EDIT MODE

Character edit mode allows the modification of the bitmap information of the selected character of the current font file. Figure 2.4 shows how this mode looks on the screen. The display consists of the following :

1) Filled polygon (Fat bit) view of the character.
2) Actual size view of the character.

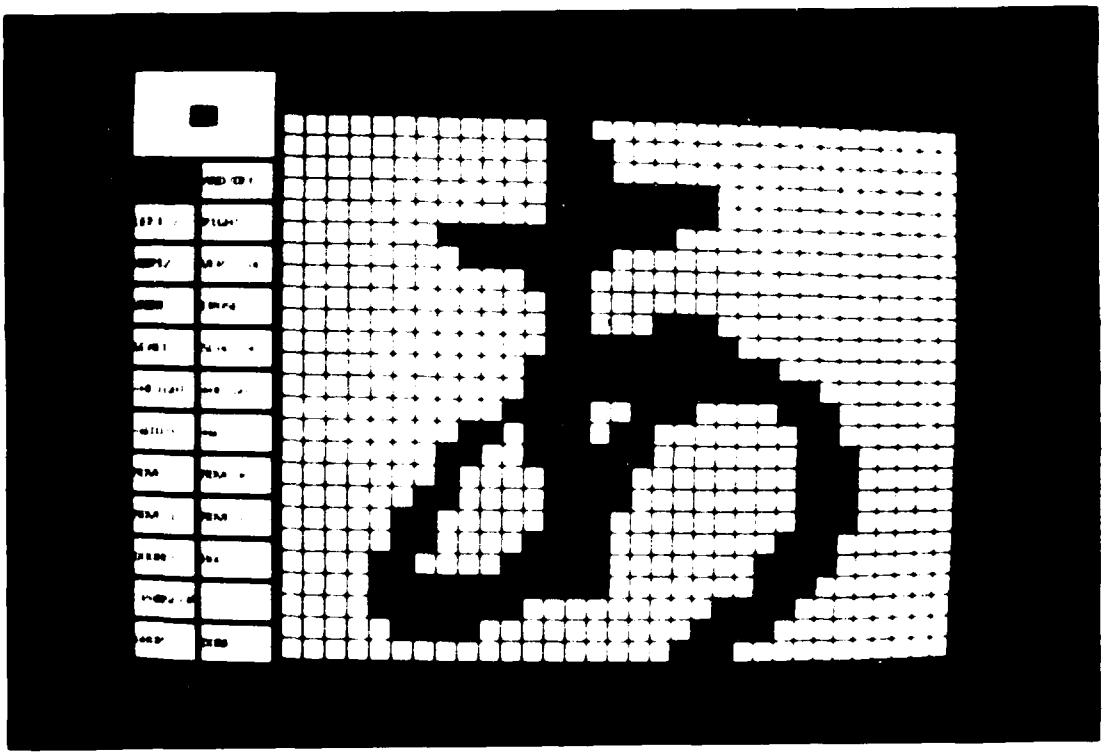


Figure 2.4 Character edit mode

3) Information on the edited character:
- Character name.
- ASCII correspondence.
- Character bitmap height.
- Character bitmap width.

4) Option area showing the editing options.

1. Pen options

Character edit mode has two pen types for the user to choose, Xor pen and Xand pen. The system pen type default is Xor.

a. Xor pen type

Xor pen type has the following behavior: If the selected fat bit is OFF [ON] then it is set to ON [OFF]. If the user moves the cursor with the mouse without releasing the button, then the fat bits that are touched by the cursor are set ON[OFF].

b. Xand pen type

Xand pen type has the following behavior: If the selected fat bit is ON [OFF] then it remains so. If the user moves the cursor with the mouse without releasing the button, then the fat bits that are touched by the cursor are set ON [OFF].

2. Line options

a. Drawing lines

To draw a line on the bitmap, the user must select first the Xor option. In this case, the Xor option enables the drawing of the lines. Then, the user must select a line type option. There are four options for the type of line to be drawn. These option are:

HORZ to draw horizontal lines.
VERT to draw vertical lines.
L. DIAG to draw left diagonals.
R. DIAG to draw right diagonals.

By clicking at the desired position inside the bitmap of fat bits, the system draws the corresponding lines. The system continues drawing lines if the user does not release the button but moves the cursor. This is desirable as a "paint mode" for the bitmap.

b. Erasing lines

To erase a line on the bitmap, the user must select first the Xand option. In this case, the Xand option acts like an erase mode. Then, the user must select a line type option. By clicking at the desired position inside the bitmap of fat bits, the system erases the corresponding lines.

3. Scaling

The fat bits can be scaled up or down to enable the user a better view of his work. This option does not alter the bitmap size, it only varies the fat bit display's size. Display scaling is performed by clicking in +SCALE or -SCALE.

4. Changing the bitmap size

The size of the bitmap can be changed by using the +HEIGHT/-HEIGHT options to increase/decrease the height of the bitmap and by using the +WIDTH/-WIDTH options to increase/decrease the width of the bitmap. The system allows heights and widths ranging from 0 to 64 bits.

5. Moving the character around

Options to move the character inside the bitmap are available. These options are: RIGHT, LEFT ,UP and DOWN. No restriction is placed on the user. It is possible to lose the character by moving it outside the bitmap.

6. Copying a character

The user can transfer the bitmap information of any character into the currently opened character. Copying a character does not erase the bitmap information of the currently opened character. The transfered character is overlayed with the existing character. This behavior is desired for constructing a character from other defined characters in the font. To copy a character, the user must select the TRANSFER option. This option displays a menu similar to the one shown by character selection mode. The user selects from this menu the character he wants to copy into the currently opened bitmap. There are two options under this mode: ABORT option and SAVE option. To transfer the selected character, the user must click in the SAVE option. Otherwise, he selects the ABORT option. Both options return the user to the currently opened bitmap.

7. Undoing the last command

The user has the option of undoing the last changes to the bitmap. To do so, the user must click inside the UNDO option in the command area. Changes to the bitmap sizes are not undone by this option. The user has to increase or decrease the size of the bitmap using the commands described above.

8. Erasing the bitmap

Each fat bit of the bitmap can be erased using the pen types described above. The user can erase the entire bitmap by using the ERASE option in the command area. This option resets all the bits of the currently opened bitmap.

9. Changing the character parameters

Editfont initializes all character parameters to avoid forcing the user to set each parameter for each character. The bitmap size is set as explained above. The other

parameters are set by clicking in the PARAMS option. The PARAMS options displays a menu showing the currently set parameters for the character. Figure 2.5 shows how this is displayed on screen.

By clicking inside the "+" or "-" areas, the PARAMS option increases or decreases the values of the parameters. The user can change three parameters in this

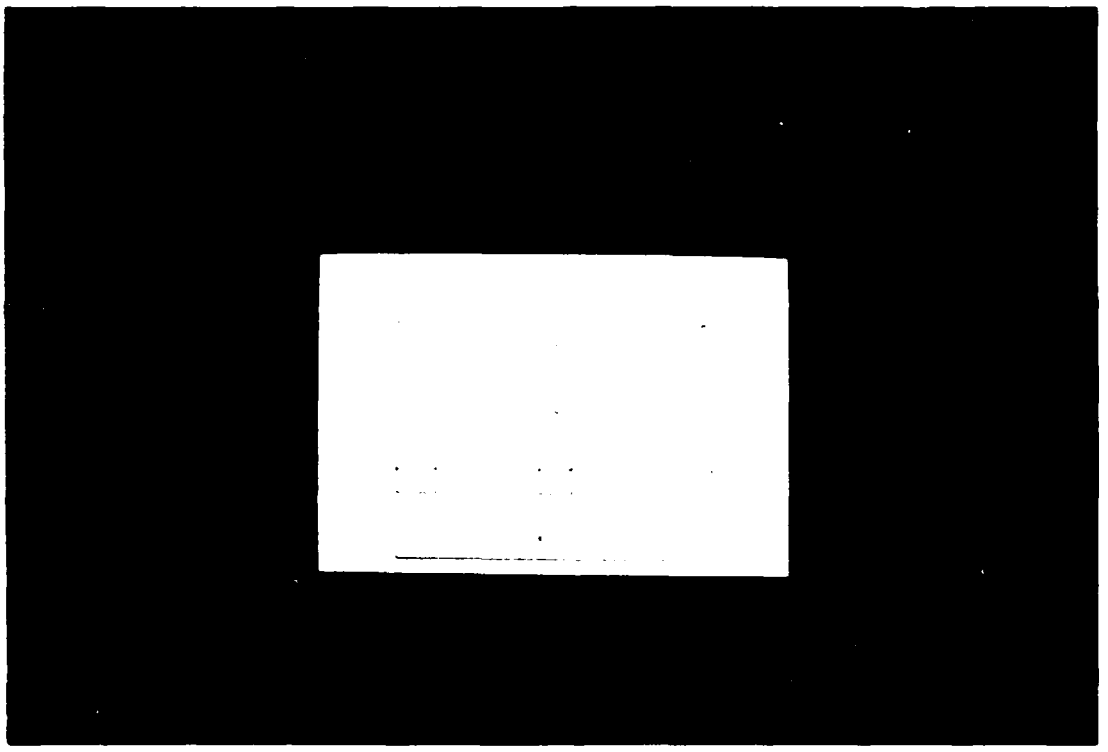


Figure 2.5 Character parameter mode

option : the X offset of the bitmap, the Y offset of the bitmap and the skipwidth. Information about character description parameters can be found in Chapter IV. The ranges accepted by the system are from -64 to 64.

A special case to be noted is the setting of the skipwidth. By default, the system sets this parameter automatically to one more than the actual character bitmap width. If the user changes the skipwidth, it no longer varies with the character width and remains constant. The only way to tell the system to set this parameter automatically is to again set its value to one more than the current bitmap width. The defaults values are as follow:

X offset = 0
Y offset = 0
Skipwidth = 1 + width of bitmap

10. Saving the character bitmap

To save the current character of the font, the user must click inside the SAVE option. If the user is not sure of the changes he has made, and does not want to save it, he can click in the ABORT option. ABORT places the system in character selection mode.

E. EDITFONT AS AN ICON EDITOR

Many graphics application programs need to use several small icons. Although **editfont** was implemented for the creation of fonts, the system editing features and the font extraction from images capability, makes this font editor a nice tool for the creation of icons.

Icons can be mapped to the ASCII characters instead of font characters. By calling the high level routines explained previously, these icons can be treated like character fonts. There is a limit when it cames to the size of the icon that can be created. **Editfont** maximum heights and widths of the character bitmap range between 0 and 64.

F. SYSTEM WARNING SIGNALS

A good user interface signals the user of the application of any errors that he has committed. **Editfont** tries to signal the user when an invalid option has been selected or something wrong has happened. The system issues a sound signal on the following conditions:

- User tries to edit a non-font file.
- When there are no more files on the directory and the user keeps scrolling.
- User types a wrong file name.
- User tries to create a new font file with the same name as an old one.
- User tries to open a non-picture file for font extraction.
- User tries to edit a character *without selecting one.*
- User exceeds the limits on sizes, limits on parameter ranges etc.

III. FONT GENERATION FROM AN IMAGE

Font generation via font extraction is a technique that enables the user to create fonts by extracting characters from a digitized image. The Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School is equipped with an Eikonix digitizer camera. This device is connected to a VAX/VMS 11/780 computer. Software for creating images with the Eikonix camera can be found in reference 1. This reference presents two useful programs. The first program called **camera,** digitizes an image using the Eikonix digitizer camera, and stores the data in a file. The second program called **display,** displays the image on the IRIS workstation.

The **editfont** font editor has the capacity for easy and rapid font extraction from an image. **Editfont** uses the same image format as the **camera** and **display** programs. The image format is explained in the last section of this chapter. The next section explains the actions necessary when using the **camera** program. These steps generate a compatible image for use in **editfont.**

A. TAKING PICTURES FOR EDITFONT

-Turn on the camera and prepare the camera according to the user instructions.

-When ready, run the **camera** program.

-At the prompt "What is the output filename", enter the desired name of the file in which the picture is going to be stored.

-At the prompt "Do you wish to do black and white or color image", enter 1. **Editfont** only accepts black and white images.

-At the prompt "Do you wish the center of your image to be the same as the center of the cross hairs on the camera", enter 1. Centered images are recommended for font extraction.

-At the prompt "Enter the number of columns wide the digitized image will be", enter 1024. This value is the maximum width resolution on the IRIS display. **Editfont** does not accept values greater than 1024. It accepts values less than this number.

-At the prompt "Enter the number of lines deep the digitized image will be" , enter 768. **Editfont** does not accept images with more than than 768 pixels of height.

-At the prompt "Enter the title of the image", the user can type any comment. **Editfont** does not use this information.

-When "push the run switch" is prompted, be sure that the camera is correctly set and push the corresponding button.

-If no error occurs, the user will have a compatible image for the use with **editfont** system.

B. USING THE PICTURE FOR FONT EXTRACTION

After taking the desired pictures with the digitizer camera, and transfering the corresponding files onto the IRIS workstation, the user has to enter **editfont** as explained in Chapter II. At the main menu, the user has to select the desired image file. Selection of the image file is done in the same way as selecting a font file. This was explained in Chapter II. When the image file name is highlighted, the user has to click in the option SELECT PICTURE. **Editfont** uses this image file when the user enters font extraction mode.

At this time, the user has to select the font file where the extracted characters are to be stored. This is done by creating a new font file or editing an old font file. By clicking in NEW or EDIT options at the main menu, the user is sent to character selection mode. Inside this mode, the user has to select the PICTURE option. This option puts the user into font extraction mode.

C. FONT EXTRACTION MODE

Font extraction mode displays the menu shown in Figure 3.1. In this mode, there are two areas: the image display area and the command area. When the cursor is moved into the image area, the arrow shaped cursor changes to a square shaped cursor with transparent interior. This cursor is used as a camera lens. The user must move this cursor to select the character for extraction. Font extraction mode contains the following display:

-Display of the selected image.
 -Cursor shown as a square for extracting the desired pixels.

-The command area contains:
 -Selection of the ASCII correspondence character
 -Manual/Automatic selector
 -Brightness/darkness selector
 -A view of the extracted character
 -Option for moving the image UP or DOWN
 -Exit option

1. Selecting the ASCII correspondence

The current ASCII correspondence character that is assigned to the extracted character is displayed inside the command area. When the user enters font extraction mode for the first time, the ASCII correspondence is set to the letter "A". There are two modes for changing the ASCII correspondence value: the MANUAL mode and the

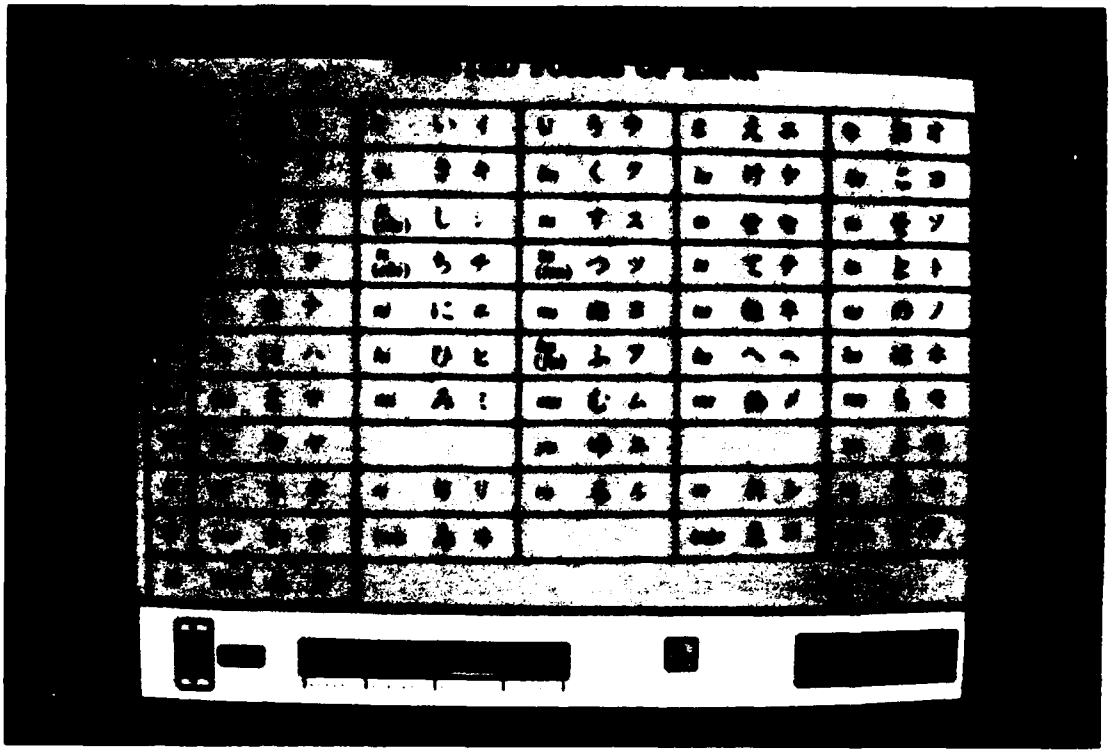


Figure 3.1 Font extraction mode

AUTOMATIC mode. In the MANUAL mode, change of the ASCII value is done by clicking into the "+" or "-" options. The "+" option increases the ASCII value and the "-" option decreases the ASCII value. For each extracted character, the user has to change this setting. In the AUTOMATIC mode, the user has to set only one time the ASCII value. Then for each extracted character, the system increases the current ASCII value to the corresponding successor value.

2. Changing the brightness of the image

Editfont has the capacity for changing the intensity of the image's brightness. This allows the user to modify the intensity cut-off point for bitmap extraction. To change the brightness of the image, the user has to move the brightness selector. This is done by placing the cursor inside the brightness selection area. Pressing the MIDDLEMOUSE and sliding the selector to the left, the brightness increases, and by sliding the selector to the right, the brightness decreases. There are 255 possible intensity values. A scale is provided for the user to refer to when changing the brightness.

3. Changing the size of the lens

To change the width of the bitmap extraction lens, the user must press the LEFTMOUSE. This action increases the width until the user releases the button. To change the height of the lens, the user must press the RIGHTMOUSE. This action increases the height until the user releases the button. The lens width and height can range from 0 to 64. If the lens width or height is increased beyond the maximum, it wraps around to zero. The default lens size is as follows:

Width = 30
Height = 30

4. Extracting a character

Extraction of a character is done by moving the lens around the image and centering the image character inside the frame of the lens. When the user is ready to extract the character, he must press the MIDDLEMOUSE. By doing this, a view of the extracted character is displayed in the command area. If the user does not agree that the character extracted is good, he can try again in the same manner. If the system is in

manual mode, then the user has to select the ASCII correspondence character for the next character to be extracted.

5. Moving the image

The font extraction mode accepts images ranging from 0 to 768 pixels high. In order to display the command area, part of the image is not seen. There are two options available to move the image display. The DOWN option displays the bottom part of the image. The UP option displays the upward part of the image.

6. Exiting font extraction mode

When the user has finished extraction of the desired characters, he must click in the EXIT option. This option returns the user to character selection mode. At this point, the user can see how his new font looks. The next step is to clean up and remodel the characters. This is done by opening the bitmap of each character. This procedure was explained in Chapter II.

D. IMAGE FILE FORMAT

Throughout **editfont,** the image file format is transparent to the user. For completeness, this section explains how the image data is handled by **editfont.**

There are three type of images that can be taken using the **camera** program: Black and white images, Color images and Dithered images. **Editfont** can only handle black and white images. The Eikonix camera has the capacity of taking images of sizes up to 4096 * 6400 pixels. This exceeds the IRIS screen resolution. For this reason, **editfont** was implemented to accept images that range from 0 to 1024 pixels wide and from 0 to 768 pixels high. Any other image size is rejected by the system. The image format used by the system consists of the following parts:

-Header
-Image pixels information

The header consists of: a two byte field for storing the type of image, a two byte field for storing the number of lines in the image, a two byte field for storing the number of columns in the image, and an eighty byte field for storing any user comment about the image. The image pixel information for a black and white image is stored as follows: (1) the image pixels are stored one by one from the upper left pixel in the first line to the bottom right pixel in the last line, (2) one byte is needed to store each pixel of the black and white image. Each pixel holds a value ranging from 0 to 255. This value indicates the gray level of the pixel. The 0 value corresponds to the white color and the 255 value corresponds to the black color.

Editfont has a default intensity value. If a pixel has a value less than the system intensity, then **editfont** extracts this pixel as white. If the pixel value is greater than the system intensity, then **editfont** considers this pixel as black. If the user changes the intensity of the system, then the system changes the color ramp to recolor all the pixels in the displayed image.

IV. FONT UTILIZATION

A. SYSTEM LIBRARY ROUTINES

Font memory is manipulated by calling font handling primitives or routines in the IRIS graphics library. There are basically four C language routines that the IRIS user can call.

1. defrasterfont

The **defrasterfont** routine defines a raster font. It loads the font data from main memory to the IRIS special font memory. The C language specification of this routine is as follows: [2]

```
defrasterfont(n, ht, nc, chars, nr, raster)
Short n, ht, nc, nr;
Fonchar chars[];
short raster[];
```

The six input parameters store the font data. This font data includes:

n	: The internal font name.
ht	: The maximum height of the characters in the font.
nc	: The number of characters in the font.
chars	: The description of each character in the font.
nr	: The size of the raster array.
raster	: An array of one dimension with index from zero to nr. This array contains the bitmap for each character.

The description of each character gives the relative position of the character bitmap with respect to the current character position. It also provides the size of the character bitmap. The **chars** array stores this description. For example, **chars**['z'].w holds the width of the character 'z'. The fields of this array are :

offset	: The position inside the array raster where the character bitmap information is stored.
w	: The width of the bitmap character.
h	: The height of the bitmap character.
xoff	: The horizontal offset from the current character position where the character is going to be displayed.
yoff	: The vertical offset from the current character position where the character is going to be displayed.
skipwidth	: The amount to be added to the current character position after drawing the character.

Figure 4.1 shows an example of a character displayed with its character description. The current character position is determined from the user call **cmov2i** or by the last string displayed. This example shows a 9 by 9 character. This size is the default character font size.

When using the **defrasterfont** routine, the user has to manipulate the font data. This means, the user has to define his own data structure, and assign the corresponding values of the font to this data structure. This work can be avoided by using higher level routines. The loading of the font data into the IRIS font memory then becomes transparent to the user. These high level routines are described below.

2. font

The **font** routine is used to select one font from the ones stored in font memory. Calls to routine **charstr** use this font selection. The C language specification is as follows:

```
font(fntnum)
short fntnum;
```

fntnum is the internal name of the font that the user wants to call.

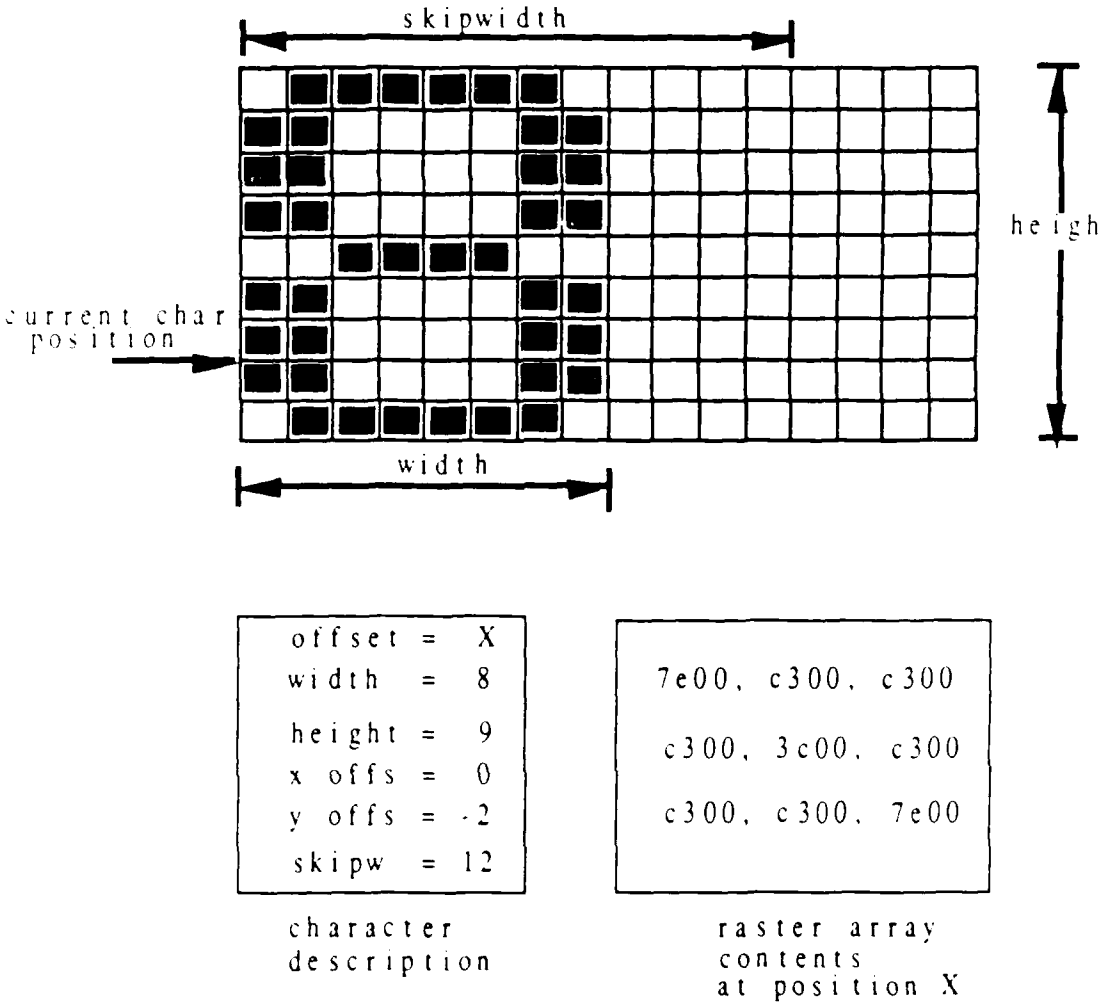


Figure 4.1 Display of a character

3. getfont

The **getfont** routine returns the number designating the font currently in use. The C language specification is as follow:

long getfont()

4. getheight

The **getheight** routine returns the maximum height value of the font currently in use. The C language specification is as follow:

```
long getheight()
```

B. HIGH LEVEL ROUTINES

There are two high level routines available for the use in any application program, **fontdef** and **delfont.** Using these routines, data structures and data transfers into font memory are transparent to the user. Appendix B shows the data structure used by these routines and by the font editor. Appendix C shows the C code of the **fontdef** and **delfont** routines.

1. fontdef

The **fontdef** routine loads a font file, created by **editfont,** from disk to the IRIS font memory. The C language specification is as follow:

```
fontdef (n, filename)
short n;
char filename[];
```

The two input parameters are:

n : The internal name that the font will take.
filename : The file that holds the desired font.

2. delfont

The **delfont** routine deletes a font from the IRIS font memory. The C language specification is as follow:

```
delfont (n)
short n;
```

The input parameters is:

n : The internal name of the font.

C. AN EXAMPLE PROGRAM

Once a font is generated using **editfont,** the user can use the font font in his application program. Fig 4.2 shows an example program that loads two different fonts into font memory, and displays two strings using these fonts.

The high level routines are stored in the file "fontdef.c". The first step is to get the portion of the code containing the high level routines. This is done by the statement **#include 'fontdef.c'**.

The loading of the different fonts is done by calling the **fontdef** routine. In the example, the font stored in the file **myfont1** is loaded into font memory using 1 as the internal name. The font stored in the file **myfont2** is loaded to font memory with the internal name 2. The 0 value is reserved for the standard IRIS font. The user can not use this internal name to define any other font. Once the desired fonts are loaded, the user can invoke these fonts at any time. In the example, **font(1)** is called first. The system uses the invoked font style when drawing text strings.

The characters are displayed using the current color definition. This can be changed by using the **color()** library routine. The example program displays two strings with different font style and different color definition.

The position where the text is going to be located on screen is set by calling the **cmov2i** routine. The position of each character with respect to the previous character is determined by the skipwidth of the previous character. The skipwidth by default is one more than the width of the character. This can easily be changed by going back to the font editor and selecting the PARAMS options inside character edit mode.

Font memory is limited by hardware configuration. After loading and using the desired fonts, these must be deleted from font memory to avoid overloading it. This is done by calling the **delfont** routine. Programs that use different fonts and textures commonly overload the font memory. If this happens, all characters are displayed using the standard font. The user must delete some textures and fonts from font memory to recover from this.

D. FONT FILE FORMAT

Editfont and the high level routines use the same file format for reading or writing the fonts. Fig 4.3 shows the bitmap information of one character stored on disk. Appendix A shows an entire font file created by **editfont** using the font extraction mode.

The first line of a font file contains the font maximum height. This value is computed by **editfont** when the user creates a font. The font maximun height value ranges from 0 to 64. Each character is stored in sequential alphabetical order. Characters not defined by the user do not occupy space in the font file. The information stored for each defined character is divided in two groups:

-The character description.
-The bitmap information.

```
#include "fontdef.c"
main()
 *
 *
 *
 *
fontdef(1,"myfont1");
fontdef(2,"myfont2");
 *
 *
 *
 *
font(1);
color(YELLOW);
cmov2i(100,400);
charstr("This is written using font 1");
 *
 *
 *
 *
font(2);
color(YELLOW);
cmov2i(700,200);
charstr("This is written using font 2");
 *
 *
 *
 *
delfont(1);
delfont(2);}
```

Figure 4.2 Example program

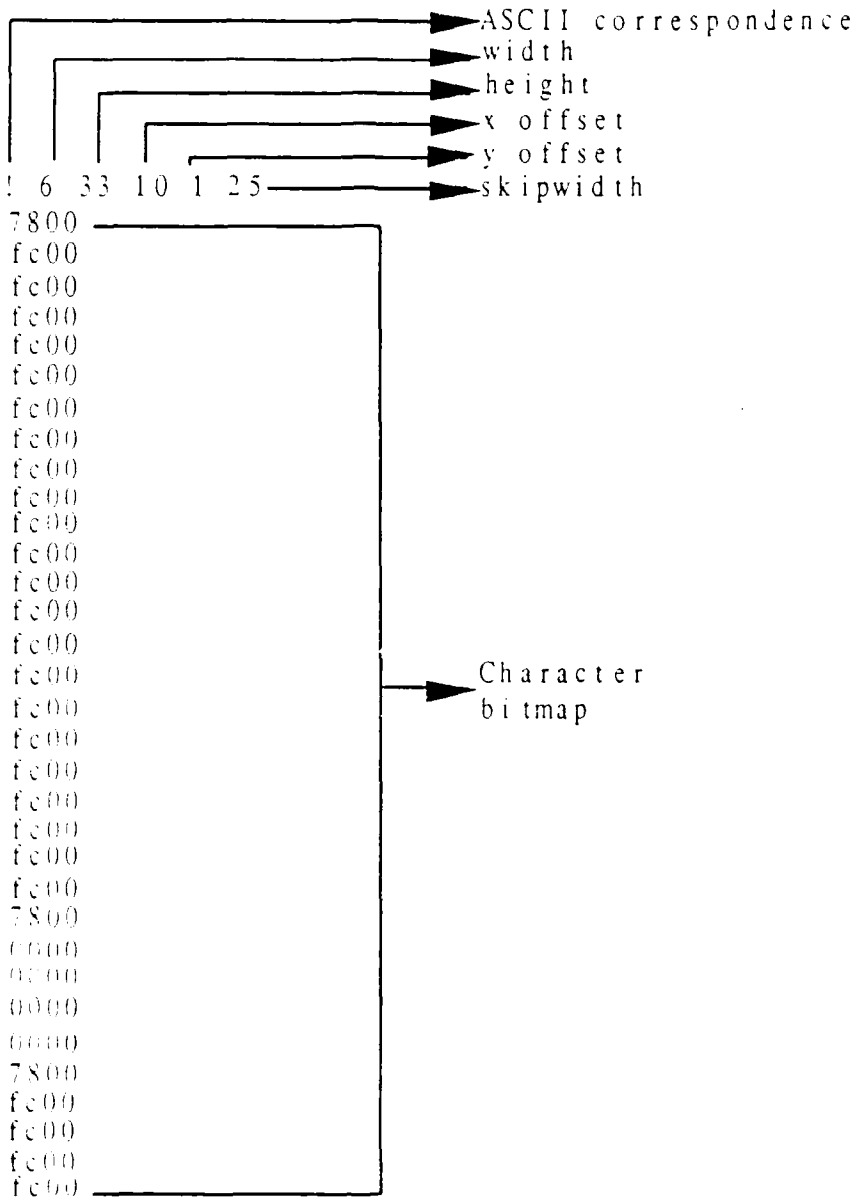


Figure 4.3 Example font file

The description of each character is stored in one record in the following order:

-ASCII correspondent
-Bitmap width
-Bitmap height
-X offset
-Y offset
-Skipwidth

The bitmap of each character is stored in several records depending on the character size. Each record contains a hexadecimal value which represent 32 bits of information taken from left to right and from top to bottom of the bitmap.

V. SYSTEM CONSIDERATIONS AND CONCLUSIONS

The **editfont** system code is comprised of 5 program files and 2 support files with a total of 3,500 lines including documentation. The code and documentation for the current version of **editfont** is available in the Naval Postgraduate School's Graphics and Video Laboratory in the Department of Computer Science.

A. SOFTWARE LIMITATIONS

The system maximum font size is 64 pixels in height and 64 pixels in width. These values have been calculated by taking the average font size needed in different graphics programs. Although the system can be changed to accept values greater than the specified above, it is not recommended as larger fonts easily overload the font memory of the IRIS.

Inside character edit mode, the user experiences some system degradation when the bitmap size is increased to its maximum. This degradation is minimal. It occurs because of the large number of filled polygons on the screen. One factor that influences this degradation is the fact that **editfont** has been implemented using double buffering. Double buffering was chosen for its capability for smooth picture transitions.

The image format used by the system is not an IRIS standard image format. The image format implemented for the system, was chosen because of its simplicity and easy data manipulation. Modification of the image format for font extraction is suggested to make **editfont** more standard.

The font extraction mode permits the user to extract different fonts from printed documents. This mode can also be used for icon extraction. In this case, the icon sizes are limited to the maximum of the font characters. The implementation of an icon editor based on the **editfont** program is recommended.

B. HARDWARE LIMITATIONS

The font editor has been implemented to run on IRIS workstations with at least 12 bit planes. **Editfont** uses somes bits of the planes to mask the character bitmap lens in font extraction mode.

Font memory is limited in storage to 16K 16 bit words. The user is at risk of overloading the font memory when he uses many fonts and textures at the same time in his application program. If the font memory is overloaded, then the text drawings are displayed using the standard font. To avoid overloading the font memory, the user must load a maximum of one or two fonts into the font memory. After using these fonts, delete them from font memory using the **delfont** routine explained previously, and then load the new fonts.

C. CONCLUSIONS AND RECOMMENDATIONS

This study has presented support tools for font generation on the IRIS graphics workstation. Graphics applications programmers can use the proposed font editor to improve the different text displays of their applications. A new technique for font generation is implemented in the proposed font editor. Font generation via font extraction. This powerful tool enables the user to generate complex fonts by extracting them from printed documents.

Many applications displays represent objects by using icons. The editing capacity of **editfont,** the font extraction feature, and the usage of the high level routines, enables the graphics application user to generate not only fonts but icons as well. These icons can be treated as font characters. The implementation of high level routines for icon handling is recommended.

Replicated fonts files are very common when different applications use the same fonts. The implementation of a font and icon library is recommended. This library should be stored in a global directory. Any user should be able to load directly from this directory the needed fonts or icons. A program should be implemented for searching the available fonts and icons stored in this global directory. This program should display any font or icons that the user wants to see. This reduces replicated data in a limited storage enviroment such as the IRIS workstation.

APPENDIX A - KANJI FONT FILE CREATED BY EDITFONT

```
36
 24 24 0 0 26
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
A 24 24 0 0 26
00000000
00300000
07c01000
00801000
3ffe1000
0081fe00
1ffc1200
10841200
10841200
1ffc1200
10841200
10841200
1ffc2200
```

00802200
00802200
0ff84200
00804200
00808200
3ffd3c00
00000000
04441000
08420800
10410400
00000000
B 24 24 0 0 26
00000000
00800000
018ffe00
03000600
02000c00
06003000
0420c000
0c230000
08630000
18c30000
fff1e000
18c01800
18dffe00
18800c00
09800c00
0d000c00
05000c00
07000c00
06000c00
07000c00
0d001800
09031800
1981f000
00000000
C 24 24 0 0 26
00000000
00c00000
00300000
00000000
01f00000
00100000
00100800
00101000
7ff02000

```
0030c000
00530000
005c0000
00980000
00940000
01140000
01120000
02110000
02108000
04104000
08102000
10101800
60100600
00f00000
00000000
D 24 24 0 0 -1
00000000
04020000
04040000
08080000
08100000
10202000
18401000
24fff800
24000400
44102200
44201000
04400800
048fc400
05104200
04284000
04448000
04830000
04030000
04048000
04084000
04102000
04201000
05c00e00
00000000
E 24 24 0 0 26
00000000
001ffe00
7fc08200
40408200
40408200
```

```
40408200
40410200
40410200
40420200
40420400
40440400
7fc80400
40401c00
40400000
405ffc00
40500400
40500400
40500400
40500400
40500400
40500400
d0044000
1ffc7f00
00000000
F 24 24 0 0 26
00000000
007c0000
01c60000
1f7ff000
10183800
30000800
07038000
0a854000
07038000
00000000
00100000
00200000
00400000
00480000
00300000
00000000
00000000
0fffc000
1aaa8000
0aa90000
0eab0000
03fe0000
00000000
00000000
G 24 24 0 0 26
ffffff00
```

```
80000100
80000100
9ffff900
90000900
90000900
93ffc900
92004900
92004900
927e4900
92524900
92724900
92024900
92024900
93fe4900
90004900
90004900
9fffc900
80000900
80000900
fffff900
00000100
00000100
ffffff00
H 24 24 0 0 26
00000000
7fcff800
40480800
40480800
40480800
7fcff800
40480800
40480800
40480800
7fcff800
40000800
40000800
40000800
40000800
40000800
40000800
40000800
40010800
40008800
40004800
40002800
40001800
```

```
00000000
00000000
I 24 24 0 0 26
ffffff00
ffffff00
c0001e00
c0003c00
c0007800
0000f000
0001e000
0003c000
00078000
000f0000
001e0000
0fff000
0fff000
00f00000
01e00000
03c00000
07800000
0f000000
1e000000
3c000300
78000300
f0000300
ffffff00
ffffff00
T 24 24 0 0 26
00000000
00000000
00000000
00000000
3e000000
08000000
08000000
0bc00000
0a400000
0a7c0000
0a500000
0bd3c000
0a924000
3e520000
02520000
0253c000
00104000
007c4000
```

```
00024000
0003c000
00000000
00000000
00000000
00000000
U 24 24 0 0 26
00000000
00000000
00ff0000
00810000
04810000
04810000
04810000
04ff0000
04000000
04000000
07ffe000
00002000
000c2000
00182000
00302000
006c2000
7ffffe00
00c60000
01830000
03018000
0600c000
0c006000
00000000
00000000
V 24 24 0 0 26
03006000
0500d000
09819800
18810800
10ff0c00
607e0400
c0000200
80000200
c387c300
4484c100
44840100
44884100
84ccc200
36478200
```

```
8ffe2a00
84fbc700
40830200
60932600
30da8600
18660400
06000400
01803800
00bfe000
00000000
W 24 24 0 0 26
00000000
00410000
00410000
01f7c000
00410000
00410000
0ffff800
00000000
00000000
01ffc000
01084000
01084000
01084000
01ffc000
01084000
01084000
01084000
01ffc000
00630000
00c18000
0180c000
03006000
00000000
00000000
Z 24 24 0 0 26
00000000
00000000
00000000
00000000
00000000
00080000
002a0000
002a0000
002a8000
002a8000
```

```
002a8000
002a8000
007f8000
06408000
07428000
03f48000
01988000
008c8000
00c68000
00438000
00430000
00420000
00420000
00420000
q 24 24 0 0 26
00000000
00000000
18000c00
14001400
12002400
11004400
10808400
10410400
10220400
10140400
10080400
10000400
10000400
10080400
10140400
10220400
10410400
10808400
11004400
12002400
14001400
18000c00
00000000
00000000
w 24 24 0 0 26
00000000
00000000
00ff0000
01008000
02004000
04002000
```

```
08001000
10000800
10000800
10000800
10000800
10000800
10000800
10000800
10000800
10000800
10000800
08001000
04002000
02004000
01008000
00ff0000
00000000
00000000
```

APPENDIX B - FONT DATA STRUCTURE

```
/* this is an IRIS-2400 Program.
  This is file fontdef.extern

  It contains the external declarations for routine fontdef so that
  other functions can access the font definition data arrays.

*/


#define MAXRASTER 16384  /* max number of raster words available */
                  /* We compute this value in the following fashion:
                    The maxwidth of each char is computed in 16 bit
                    words. That value is multiplied by the maxheight.
                    That value is then multiplied by 128 chars in the
                    set. For example, 48 bit by 48 bit chars need
                    18432 raster spots. 64 bit by 64 bit chars need
                    32768 raster spots.

extern Fontchar chars[128];     /* the Font table */

extern unsigned short raster[MAXRASTER]; /* the raster defs for this font      */

extern long maxheight;          /* the max pixel height for this font  */

extern long maxwidth;           /* the max pixel width for this font */

extern long chardefined[128];   /* TRUE if the char is defined, FALSE
                          otherwise */

extern long rptr;               /* the last written spot in array raster */

extern unsigned long temp[1000]; /* this array is used to reverse the char
                           defs. It must equal maxwidth in 16 bit
                           words times maxheight. 256 is good for
                           max 64 by 64 chars.
                         */
```

```
/* this is an IRIS-2400 program */
/* this is routine fondef

  It defines a new raster character font.
  It reads a specified font from a font file.
*/

#include <stdio.h>
#include "gl.h"

/* get the declarations for the font */
#include "fontdef.h"

fontdef(fontnum,filename)

/* you select the number you want to call this guy */
long fontnum;

/* passed in file name */
char filename[];
{

  /* temp loop index */
  long i,j,k;

  /* temp loop variable */
  long jj;

  /* file pointer for the font file */
  FILE *rfp;

  /* temp char value */
  char charval;

  /* size of this bitmap (real size) */
  long width,height;

  /* xoffset and yoffset for the char*/
  long xoffset,yoffset;
```

```
/* amount to skip after this char is in */
long skipwidth;

/* temp char array */
char tmp[150];

/* number of words per row */
long words;

/* temp counter to read in the bitmaps */
long icnt;

/* clear the char table... */
for(i=0; i < 128; i=i+1)
{
  /* no space for this char def */
  chars[i].offset=0;

  /* bitmap is zero in width */
  chars[i].w=0;

  /* bitmap is zero in height */
  chars[i].h=0;

  /* no x offset */
  chars[i].xoff=0;

  /* no y offset */
  chars[i].yoff=0;

  /* no skip width */
  chars[i].width=0;

  chardefined[i] = FALSE;

}

/* clear the raster array */
for(i=0; i < MAXRASTER; i=i+1)
{
  raster[i]=0;
}

/* no max width yet... */
maxwidth=0;
```

```
/* open the named font file */
rfp=fopen(filename,"r");

if(rfp == NULL)
{
  perror("FONTDEF:");
  printf("FONTDEF: cannot open file %s!0,filename);
  exit(1);
}

/* read the max height in pixels */
fscanf(rfp,"%d",&maxheight);

/* scan past the end of the line */
fgets(tmp,150,rfp);

/* say that we havent used any raster space yet */
rptr = -1;

/* read until we run out of file */
while(TRUE)
{
  /* get a char def line */
  i=fscanf(rfp,"%c %d %d %d %d %d",&charval,&width,&height,
        &xoffset,&yoffset,&skipwidth);

  /* scan past the end of the line */
  fgets(tmp,150,rfp);


  if(i <= 0)
  {
    /* eof */
    break;
  }

  /* we have a character def... */

  /* do we have a new max width? */
  if(width > maxwidth)
  {
    maxwidth=width;
  }

  /* we have a character def */
  j=charval;
```

```
/* say the char spot is defined */
chardefined[j]= TRUE;

chars[j].offset=rptr+1;  /* start of this bit map */

chars[j].w=width;        /* width of this bitmap */

chars[j].h=height;       /* height of this bit map */

chars[j].xoff=xoffset;   /* x offset for the char */

chars[j].yoff=yoffset;   /* y offset for the char */

chars[j].width=skipwidth; /* skip this many pixels after
                     you draw the char */

/* we need to read 'height' rows of data.
  the first row we read is the last one to go into
  array raster.
*/
/* compute number of words per row */
words = ((width-1)/16)+1;

/* the total space we need is i times height */
i=words*height;

/* read each row... */
icnt = -1;

for(k=0; k < height; k=k+1)
{

  icnt=icnt+1;

  /* we read across the row but its backwards... */
  for(jj=1; jj <= words; jj=jj+1)
  {
    fscanf(rfp,"%4x",&temp[icnt+words-jj]);
  }

  icnt=icnt+words-1;

  /* skip past end of line */
  fgets(tmp,150,rfp);
}
```

```
    /* reverse the values in the temp array */
    for(k=i-1; k >= 0; k=k-1)
    {
      rptr=rptr+1;
      raster[rptr]=temp[k];
    }

  }  /* end while there are char defs in the file */

  /* check to see if we wrote past the end of raster .. */
  if(rptr >= MAXRASTER)
  {
    printf("FONTDEF. We have written beyond the end of array raster!0);
    exit(1);
  }

  /* call routine to set up raster font definition */
  /* fontnum = the font number to use to call up this font.
     maxheight = the max height in pixels of characters in this font.
     128 = the number of characters in this font.
     chars = the character table.
     rptr+1 = the number of words in array raster.
     raster = the bit maps for the chars.
  */
  defrasterfont(fontnum,maxheight,128,chars,rptr+1,raster);

  fclose(rfp);

}
/* this is an IRIS-2400 program */
/* this is routine delfont

  It deletes a font from font memory */

delfont (fontnum)
/* internal font name */
long fontnum;
{
  defrasterfont(fontnum,0,0,chars,0,raster);
}
```

LIST OF REFERENCES

[1] Sando, Jean M. and Wetherald, Thomas S., "Using the Eikonix Digitizer camera with th IRIS Graphics Workstation," Technical Report, NPS-52-87-038, Naval Postgraduate School, Monterey, California, February 1985.

[2] "IRIS User's Guide," Document number 007-1101-030, Silicon Graphics, Inc., Mountain View, California, 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
4.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Michael J. Zyda, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
6.	Professor C. Thomas Wu, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
7.	Division de Informatica, Dpto. Sistemas Embassy of Peru Naval Attache Office 2160 Wisconsin Ave. N.W Washington, D.C., 20007	1
8.	Hector Mariscal O. Urb. Canupus Lt. 1 Mz. c Surco, Lima 33 LIMA-PERU	2

END

DATE

FILMED

5-88

DTIC