To the Graduate Council:

I am submitting herewith a thesis written by Jack Chung-Chih Wang entitled "Rasdazl: An Extended Graphics Tool for use with Rascal". I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

J. Michael Moshell, Major Professor

We have read this thesis
and recommend its acceptance:

David Matuszek

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

Thesis
81
.W525
Cop.2

RASDAZL: AN EXTENDED GRAPHICS TOOL FOR USE WITH RASCAL

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Jack Chung-Chih Wang

December 1981

3055644

ACKNOWLEDGMENTS

The author wishes to express his sincere appreciation to Dr. Mike Moshell for his assistance, guidance, and encouragement in the development of this thesis. Appreciation is also expressed to Dr. David Matuszek and Dr. David Straight for their support as committee members.

I would especially like to thank my wife, Ellen Shiah-Ling Wang, for her dedication, support, and typing in preparing this thesis. Without her great help, the accomplishment of this thesis would have been much more difficult.

Special gratitude is extended to my family for their understanding, encouragement, and support. To my father-in-law, mother-in-law, and their family I owe the most thanks for their careful taking care of my child while I studied my Master's degree. I would also like to thank them for their encouragement and support.

ABSTRACT

Many useful graphics programming tools are possible with the Apple II microcomputer, but were not included in the Rascal animation system because of space and time limitations. This project reports on the construction of two graphics libraries and two graphics demonstration programs.

One of these libraries provides for character or text string manipulation in the Apple II high-resolution graphics area. A set of special WRITE and WRITELN commands in Pascal formate are used to write a character, a text string, or an integer in a "text window".

The other library is built for "field" or picture buffer manipulation and high-resolution "solid image" manipulation.

To demonstrate the effects of these two libraries, two "super-painter" programs (TEXT and FIELD) are used.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Apple II Graphics Computer

The Apple II microcomputer from The Apple Computer Inc., is the machine that was used to accomplish this project. It is worthwhile to introduce some of its hardware and software features before further discussion.

There is one text mode and there are four different kinds of graphics modes in the Apple II system. The four graphics modes are low-resolution graphics mode, high-resolution graphics mode, mixed text/low-resolution graphics mode, and mixed text/high-resolution graphics mode. In text and low-resolution graphics modes, an area of memory containing 1024 bytes is used as the source of the screen information. Text and low-resolution graphics share this memory area. In high-resolution graphics mode, two large areas (8192 bytes each) are used to hold the screen information. They are called the "picture buffers" (In the Rascal system, they are called foreground and background buffers). To set the Apple into one of the above modes (text or graphics modes), a program needs only to refer to the addresses of the memory locations which correspond to the switches that set that mode. (To see these screen switches, please refer to the Apple II Reference Manual,

p.13). Machine language programs should use the hexadecimal addresses for these memory locations; BASIC programs should PEEK or POKE their decimal equivalents; Pascal programs should contain a special function or procedure to handle the jobs of PEEK or POKE done in BASIC.

1.2 Review of Other Graphics Computers

In the computer market, graphics computers can be grossly classified into 2 levels according to their selling price. Low-cost graphics computers are those priced in the range of $200 to $3000. These computers are the Atari 400/800, Radio Shack TRS-80, PET 2001, Sinclair ZX80, Commodore VIC20, Apple II, TI-99/4, and the IBM personal computer. High-cost graphics computers are graphics-oriented computers in the $5000 to $7000 price bracket. These are the Apple III, HP (Hewlett-Packard) 2648, Data General Enterprise 1000, and the Tektronix 4051 and 4027. Since these computers are far beyond most people's budgets, such computers are mostly used by larger institutions, groups, and clubs.

Because we are only interested in the Apple II and its closest competitors, what follows are brief introductions to the Atari 800, TI 99/4, TRS-80 and IBM personal computers.

Atari 800. The Atari computers from Atari Inc. are sort of a mixture of the best of the Apple II, PET 2001, and the TI-99/4. Color control allows up to four colors at the same time and eight brightness levels for each color. Like

the Apple II and TI, Atari offers plug-in ROM cartridges for languages and for application programs. The interrupt feature is directly available to the programmer, unlike the Apple II. (The software of the Apple II system makes little use of the interrupt system of the 6502 microprocessor. However, it provides several facilities to aid the programmer in the use of interrupts.) One of the most important features in the Atari is that the screen RAM (picture buffer) can be located anywhere in the address space of the computer and moved around while the program is running, whereas the other machines (such as Apple II) use fixed-screen memory areas. The graphics resolution in the Atari is 320 dots wide by 192 dots high. Atari uses the 6502A microprocessor (the same as Apple II) and its RAM can be expanded to 48 K.

TI-99/4. The TI-99/4 from Texas Instruments Inc. is somewhat of a disappointment from a graphics programmers's standpoint. The high-resolution 256 dots wide and 192 dots high screen matrix is not directly accessible through BASIC (this has been remedied in the TI-99/4A), although characters on an 8 x 8 grid may by created and stored in RAM. TI does not allow access to machine language and no PEEK or POKE statements are provided. TI's plan is to sell programmed ROMs - solid-state software - which are complete application programs written in machine language by TI programmers. These do not use the high-resolution graphics.

TRS-80 Model I. The color available in the TRS-80 from Radio Shack Corp. is black and white "monochrome" and its graphics systems can show an array of blocks 128 wide and 48 high. The TRS-80 allocates a specific 1K bytes of RAM as screen memory. The data in this area is interpreted as characters (using a character set in ROM) before display on the screen. A new version of TRS-80 is called the TRS-80 Color computer. The key difference between the Color computer and the monochrome TRS-80 is that the Color computer uses the Motorola 6809E microprocesser while the TRS-80 uses the Z80. The TRS-80 Color computer has five distinct graphics modes available; two low-resolution, two medium-resolution and one high-resolution. The low- and medium-resolution modes each offer a choice of two-color or four-color modes. The high-resolution mode has only the two-color mode. Basically, the data in memory are interpreted as pixels in these graphics modes. In the two-color modes, each bit is mapped one-to-one on the screen. If the bit is set, the pixel is colored; if not, it is black. There are nine colors available in the TRS-80 Color computer. The Color computer also provides paddle controls and interrupt features.

IBM Personal Computers. IBM (International Business Machines Corp) introduced its keenly anticipated IBM personal computer in August, 1981. Based on Intel Corporation's 16-bit 8088 microprocessor, the new graphics

computer is slated to appear in the market by October 1981. The low price and consumer-oriented design make it a head-to-head competitor of personal computers like Apple II, TI-99/4 and TRS-80 Color.

The operating system was developed by Microsoft Inc., and is called "IBM personal computer DOS". IBM intends to offer the UCSD P-System and CP/M-86 OS in the near future. With a color monitor, the IBM Personal Computer will support 16 foreground and eight background colors. In the medium-resolution graphics mode, screen resolution is 320 dots wide and 200 dots high. In the high-resolution graphics mode, resolution is 640 by 200 dots. Text and graphics can be mixed, allowing you to label items in a graphics display.

The entry-level system includes 40K bytes of ROM and 16K bytes of user RAM memory. At extra cost, the user memory can be expanded to 256K.

1.3 Rasdaz1

The Rascal system is a cartoon animation library for Apple/UCSD Pascal. This project developed a graphics system called Rasdaz1 to supplement the Rascal system, together with two "super-painter" programs (TEXT and FIELD) which demonstrate the libraries and make it possible to build sophisticated visual images. This project consisted of two major tasks. One task was to manipulate text strings in the high-resolution graphics area. WCHAR which writes a character in the graphics area and WSTRING which writes a

string in the graphics area are two basic procedures in this task. Graphics write statements is a set of WRITE and WRITELN commands (procedures) in Pascal format which allows the user to write a letter (SCWRITE), a text string (SWRITE or SWRITELN), or an integer (IWRITE or IWRITELN) to the Apple II high-resolution graphics area in a given text window.

"Typewriter" is one of the interactive procedures in the TEXT demonstration program. It allows the user to type a text string on the screen by keyboard entry when the Apple is in the graphics modes (either mixed text/graphics or all graphics mode). It is a "terminal" like typewriter. A blinking cursor that comes up with the typewriter is used to position a text string on the screen and a rubout key allows the user to erase a miss-typed letter. Text scrolling is also provided for unrotated characters (EAST case). Besides the terminal characteristics, typewriter can produce double-sized letters, solid colored letters, video inversed letters, and four different kinds of orientation letters.

The other task in the Rasdazl project is "field" manipulation. A "field" is defined as the contents of a picture buffer (an 8192-byte area of memory). Field manipulation allows the user to create up to four fields, copy any one field to another, combine two fields with pixel-by-pixel "OR" and add selective parts of one field to another field by two techniques called "wiping" and "piping".

Background and foreground are two picture buffers that are used in the Rascal system. In Rasdazl, foreground is one of the 4 fields and is always displayed on the screen. Background is used as a temporary buffer and can be displayed on the screen by using a particular command.

There are 48K bytes of RAM in the Apple II system, plus 16K bytes on the language card. Some of this is reserved for use by the System Monitor, UCSD Pascal system, and system functions. The Rascal system uses about 9K bytes of memory. Thus, the actual space that can be used for programs is no more than 30K bytes. It is impossible to use this 30K for programming as well as creating four fields of 8K bytes each. Furthermore, an extra 8K bytes (background) is necessary for a temporary buffer for field manipulation. Therefore, disk storage is used as a "virtual memory" to overcome this problem.

One of the important techniques in field manipulation is dragging ("piping") a region of a given field (called a "solid image") up, down, right or left. Since this is done interactively, when the image is moved the user has the choice of repairing or not repairing the pre-existing scene (i.e., not damaging or damaging the pre-existing scene while the user is dragging the solid image across it).

The idea of this project is to "import" to the Apple II several nice features that are found in other systems or software packages, such as colored text, orientable characters, and double-sized characters. In the Commodore

VIC20, colored characters (either in uppercase or lowercase) can be displayed on the screen by keyboard entry. However, since all the jobs (character screen display) are done by the hardware, it is very difficult for the programmer to implement features such as double-sized or orientable characters. By comparison, the Atari 400/800 is more flexible in software than the Commodore VIC20. Like Apple II, to produce a solid color, the Atari requires a corresponding bit pair for each byte. (For instance, 01 for gold, 10 for green, and 11 for blue) Therefore, the character set that is used to produce colored characters in the Atari is defined in a way that the number of "1" bit of each byte (horizontal row of pixels) in a character set would be a power of 2. Figure 1-1 shows the binary representation of the character A in the Atari character set.

```
00000000
00011000
00111100
01100110
01100110
01111110
01100110
00000000
```

Figure 1-1. Character A in Atari Character Set

Both the Apple and Atari have a software package to allow the user to define his own character set and store it in the RAM area. The difference is that the Atari requires that this character set be located on a 512-byte boundary,

while there is no limitation for the Apple II. A less-than-flexible software feature in the Atari is that the color producing and up side-down characters (called WEST orientation in this project) are done by the hardware. This makes it more difficult to implement features such as double-sized characters, the other orientations (such as NORTH and SOUTH in this project), and text scrolling in the graphics area.

Colored characters are not directly accessible to the user in the TRS-80 Color computer. However, it is possible to implement them in software. The high-resolution pixels in the TI-99/4 are not accessible through BASIC, although characters on an 8 x 8 matrix may be created and stored in RAM. PEEK and POKE are also not provided in TI-99/4. In short, the software manipulation of text is not allowed in the TI-99/4.

One package which is similar to this project is the Keyboard Filter from Mountain Hardware inc. Several nice features of Rasdazl can be found in the Keyboard Filter, such as colored characters, orientable characters, double-sized characters, and text scrolling in the Apple II high-resolution area. However, there are still several differences between them from the user's standpoint.

1. Extra hardware equipment and installation are needed for the Keyboard Filter: The Keyboard Filter is a 2 K ROM chip and can only be used with ROMPLUS+ (an Apple memory expansion board from Mountain Hardware inc.).

2. Usage: In the Keyboard Filter, all the features can be invoked only by keyboard entry while they can be invoked either by calling an appropriate procedure or through keyboard entry with an interactive program (such as TEXT) in Rasdazl.

3. Software: The text scrolling in the Keyboard Filter is performed on the entire screen (scene, text, and all are moved up one character line). In Rasdazl, the user can set his own text window and the text scrolls within the given text window only.

The other features of Rasdazl such as field combination ("OR" the images in the foreground and background), wiping (copy a region of background scene into its corresponding foreground), and piping are nice features for the Apple II computer and are not found in the other systems or packages.

CHAPTER II

LIBRARIES OF RASDAZL

There are three libraries that are used to accomplish Rasdazl. They are FREEKEY, WSTRLIB and FLDLIB. FREEKEY is the library that combines the KEYJOY and RASIO libraries in the Rascal system and the procedures which are used to support WSTRLIB and FLDLIB. WSTRLIB is a library for text manipulation and FLDLIB is used for "field" manipulation.

The following discussion are only those procedures that can be used by the user. The interactive procedures will be discussed in the next chapter.

2.1 FREEKEY - Supporting Procedures

This section only discusses the supporting procedures for WSTRLIB and FLDLIB. If you are interested in any KEYJOY procedure's description, please refer to "Rascal and Interpas Reference Manual" from Gentleware Corp, 1981.

FREEKEY introduces two data types: GROUND and CHARSET. TYPE GROUND consists of constants FOREGROUND, BACKGROUND, and BFGROUND (back- and foreground). CHARSET is defined as a set of characters.

We now examine the supporting procedures of FREEKEY.

RASCALIN. This procedure loads the background version of the Rascal system into the memory. Before any Rasdazl procedures can be used, RASCALIN must be invoked. This is done by placing RASCALIN as the first statement of any Rasdazl main program. The background version of the Rascal system is loaded from the disk file named "RASBACK.OBJ". Thus, you must place this file on your disk in disk drive 1 (volume 4) before this procedure is called. The call RASCALIN has the same effect as the call RASCALIN(BACKGROUND) in the Rascal system.

DISPLAY1. This procedure displays the foreground (page 1 of the Apple II high-resolution graphics pages) on the screen in mixed text/graphics mode or all graphics mode. If the global variable MIXMODE is set, then the call DISPLAY1 would display the screen in the mixed text/graphics mode, otherwise in all graphics mode. A global integer variable TOPSCRN (top screen) is affected when this procedure is invoked. The value of TOPSCRN is set to 159 if this procedure is called with MIXMODE set, else it is set to 191.

DISPLAY2. This procedure works the same as DISPLAY1 except it displays the background instead of foreground.

READKEY(VAR response:CHAR;accepted:CHARSET). Frequently it is necessary to ask for a character input, while continuing to move the stick under keypad and game paddle control. READKEY maintains stick motion while waiting for

the user to press one of the keys named in the given set of characters. The accepted character is returned via 'response'.

FILLBLOCK(low-x, high-x, low-y, high-y, color, ground). This procedure fills a "viewport" with the given screencolor. The viewport is defined by x and y limits specified by low-x, high-x, low-y, and high-y. This viewport can be in FOREGROUND, BACKGROUND or BFGROUND which is indicated by the argument 'ground'.

COPYBACK. This procedure is called to copy the foreground scene to the background.

2.2 WSTRLIB - Procedures for Text Manipulation

To use this set of procedures, you must place the statement

```
USES RASCAL,APPLESTUFF,FREEKEY,WSTRLIB;
```

after the PROGRAM statement, and before any other statement in the main program.

One data type is used in this library. ORTYPE consists of EAST, WEST, NORTH, and SOUTH which represent four kinds of orientation.

If one copies characters from the UCSD SYSTEM.CHARSET file (see chapter IV) directly into the high-resolution area, the displayed characters are not uniformly white. We refer these as "multicolor" characters; they are sometimes

useful when colored characters are too blurred or too wide for a particular application.

SETCMD(double,inverse,scroll,ortype,screencolor). This procedures defines the way characters get written on the screen. Double, inverse, and scroll are declared as boolean variables. For instance, a statement SETCMD(TRUE,FALSE,FALSE,WEST,WSTRCOLOR); would set the characters in double-sized, WEST orientation with the previous character color for WCHAR, WSTRING, and the graphics write statements. WSTRCOLOR is a global variable which holds the latest character color and is used to prevent the characters color from being disturbed in this case. If 'screencolor' had been BLUE, the next text output would be blue, and WSTRCOLOR would also be set to BLUE.

RESETCMD. This procedure resets the way characters get written in the graphics area to the default mode. The default mode is single-sized, none-inverse, none-scrolling, and multicolor characters.

BOTTOMSET. This procedure initializes a "text window" at the bottom of the graphics area for graphics write statements (SWRITE, SWRITELN, IWRITE, IWRITELN AND SCWRITE).

GETWINDOW(low-x, high-x, low-y, high-y). By invoking this procedure, a text window for graphics write statements can be relocated to a place other than the bottom of the graphics area. The text window is established by four

arguments. If either low limit equals or exceeds its corresponding high-limit, the text window is not changed. If any limit falls outside the range (0, 279, 0, TOPSCRN), then the range value is used instead of the given value. For instance, the call GETWINDOW (0, 30, 20, 170) would have the same effect as GETWINDOW (0, 30, 20, 159) (supposing that TOPSCRN=159).

CGOTOXY(x,y). This procedure sends the graphics write statement to the coordinates specified by x and y. The lower-left corner of the graphics area is assumed to be (0, 0). x and y must be in the pre-set text window limits. If x and y fall outside the text window, this procedure is not effected. If either x or y falls outside the window, then the lower range value of its corresponding coordinate will be used. For instance, suppose the text window is set to (10, 100, 20, 90), the call CGOTOXY (110,40), would have the same effect as CGOTOXY (10, 40).

The following procedures use a file SYSTEM.CHARSET to put a character or string on the screen. You must place this file on the disk in disk drive 1 (#4) before you use these procedures.

WCHAR(x, y, char). This procedure places the single character specified by 'char' on the screen at the location specified by x and y coordinates. For instance, to put an x in the center of the screen, you may call WCHAR(137,90,'X');

x must be in the range 0 through 279; y must be in the range 0 through TOPSCRN. This procedure does nothing, if either x or y falls outside its range.

WSTRING(x, y, string). This procedure places the string in the graphics area at the location specified by x and y coordinates. If the string is too long and runs off the screen (which side of screen depends on what orientation character is being used), the output appears back to the left at the position at which previous string started. The range of x and y are the same as WCHAR.

The following procedures mimic Pascal WRITE and WRITELN statements. Therefore, they operate in the EAST orientation only, and text scrolling is always provided. (i.e., these two characteristics can not be changed by SETCMD).

SWRITE(string). This procedure writes the string in the text window which is either set by default (at the bottom of the graphics area) or by GETWINDOW. The string starts at the location where the previous graphics WRITE statement (SWRITE, IWRITE or SCWRITE) ended in the text window or at the left window margin if the previous statement was a write line prodecure (SWRITELN, IWRITELN). For example, the following commands would produce the text in Figure 2.1.

```
SWRITE('THE FIRST SWRITE');
SWRITE('** THE SECOND SWRITE');
SWRITELN('WRITE LINE');
IWRITE(100);
```

SWRITELN(string). This procedure has the same effect as SWRITE except that it ends the current text line.

SCWRITE(char). This procedure has the same effect as SWRITE except that it writes a single character in the text window.

IWRITE(ival). This procedure has the same effect as SWRITE except that it writes an interger value specified by 'ival' in the text window.

IWRITELN(ival). This procedure has the same effect as IWRITE except that it ends the current text line.

CWRITELN(string). This procedure calls either Pascal WRITELN or SWRITELN. If the global variable SPWRT is set, then it calls SWRITELN, else it calls Pascal WRITELN.

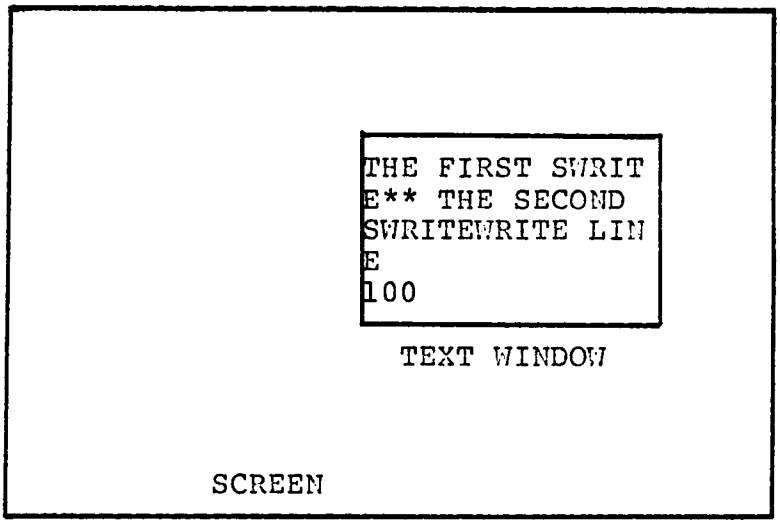


Figure 2-1. Graphics WRITE and WRITELN in the Text Window

2.3 FLDLIB - Procedures for Field Manipulation

To use this set of procedures, you must place the following statement as the first statement of the main program.

```
USES RASCAL,APPLESTUFF,FREEKEY,FLDLIB;
```

Two data types, IOTYPE and SOLIDFIG are used in this library. IOTYPE is defined as (RD, WT) where RD represents read and WT represents write. SOLIDFIG is used to define solid images and will be discussed more detail in chapter IV.

FILEIO(filename,loc,blockno,mode). This procedure is used for reading, writing and opening a disk file. The prefix and suffix of a file name default to "#4" and ".SCN". Since input and output are dealing with background and foreground in this library, the location ('loc') should be either 8192 or 16384. 'Blockno' is the number of blocks you want to read or write and 'mode' is defined as IOTYPE. Any unsuccessful I/O will give an appropriate error message.

GETFLD(filename,mode). 'Mode' is a parameter of type GROUND. This procedure loads a scene into the FOREGROUND, BACKGROUND or BFGROUND depending on 'mode'.

SAVEFLD(filename). This procedure is invoked to write the foreground scene to the disk with the 'filename'.

SWITCHF(filename). This procedure is invoked to switch the scene named 'filename' on the disk and the foreground scene. After SWITCHF is called, the scene on the disk is loaded into the foreground and the foreground scene is save on the disk with the 'filename'.

COMBF(TOPSCRN). This procedure combines the scene in the foreground and the scene in the background by "OR"ing their images. The result scene is placed on the foreground. This is an external (assembly) procedure.

SETSFIG(low-x,high-x,low-y,high-y). This procedure sets the data (such as XLOC, YLOC, XLEN, YLEN) for the source solid image ("SFIG") with the four arguments. The low-x and low-y defines the lower-left corner of the image.

SETDFIG(x,y). This procedure sets the image data for the destination image ("DFIG"). The lower-left corner of DFIG is defined by x and y. Its XLEN and YLEN are copied from SFIG.

WIPEOUT(x,y,x-length,y-length). This procedure copies a region of scene in background into its correspounding foreground with four arguments; x and y gives the lower-left corner of this region and x-length and y-length gives its size. This is an exteral procedure.

PIPEACT(x-dest,x-source,x-length,y-length,page). When this procedure is invoked, a region of scene (in foreground or background) defined by x-source, y-source, x-length and y-length would be copied into the foreground at the location specified by x-dest and y-dest. The source region of scene is in foreground if 'page' is 1 or in background if 'page' is 2. This is an external procedure.

Appendixes contains a quick reference list of the procedures in these three library UNITs.

CHAPTER III

USER'S MANUAL

3.1 Overview

To make effective use of the two super-painter programs (TEXT and FIELD), this chapter introduces the general idea of their operation. All commands are invoked by single key strokes. For most commands, the key that is used to activate the command does not represent the first letter of the phrase that terms that command. This is because two keypads (keys around K and D) have been used to control a movable stick, and because two command phrases often have an identical first letter. Therefore, the user should invoke the command appropriately according to the menus provided.

There are two demonstration programs called TEXT and FIELD in this project. Once a program is executed, it enters a main state from which the user can select a command (sub-state) according to the main menu shown on the bottom of the screen. From many sub-states, the user can enter another sub-state (called a second sub-state). For simplicity and consistency with UCSD Pascal, a control-C (hold the key marked CTRL and type C simultaneously) exits every second sub-state and returns the user to a sub-state; the Q key exits the sub-state and returns the user to the main state. In the main state, the Q key quits the main

program if the user responds with a "Y" to the assurance question.

Figures 3-1, 3-2 and 3-3 show the state diagram for TEXT and FIELD. All states marked  mean that a stick (a movable figure defined in the Rascal system) whose color, position, angle, and size are under appropriate key control is available. All states marked  mean two sticks are available and movable.

3.2 Stick Control

In some of the states, a stick is available to the user to draw a scene on the screen. In piping mode, the user can move two sticks to specify where to move images. In addition, some applications of Rasdazl are also implicitly or explicitly dealing with stick control. The following discuss these controls.

PADDLE 0. If Button(0) is pushed, the stick is painted on the scene. The orientation of the stick is set by the value of Paddle(0). In most circumstances, the angle of the stick can be controlled in the range of 0 to 360 degrees. In piping mode, the angle of the sticks is set internally to range from 0 to 90 degrees.

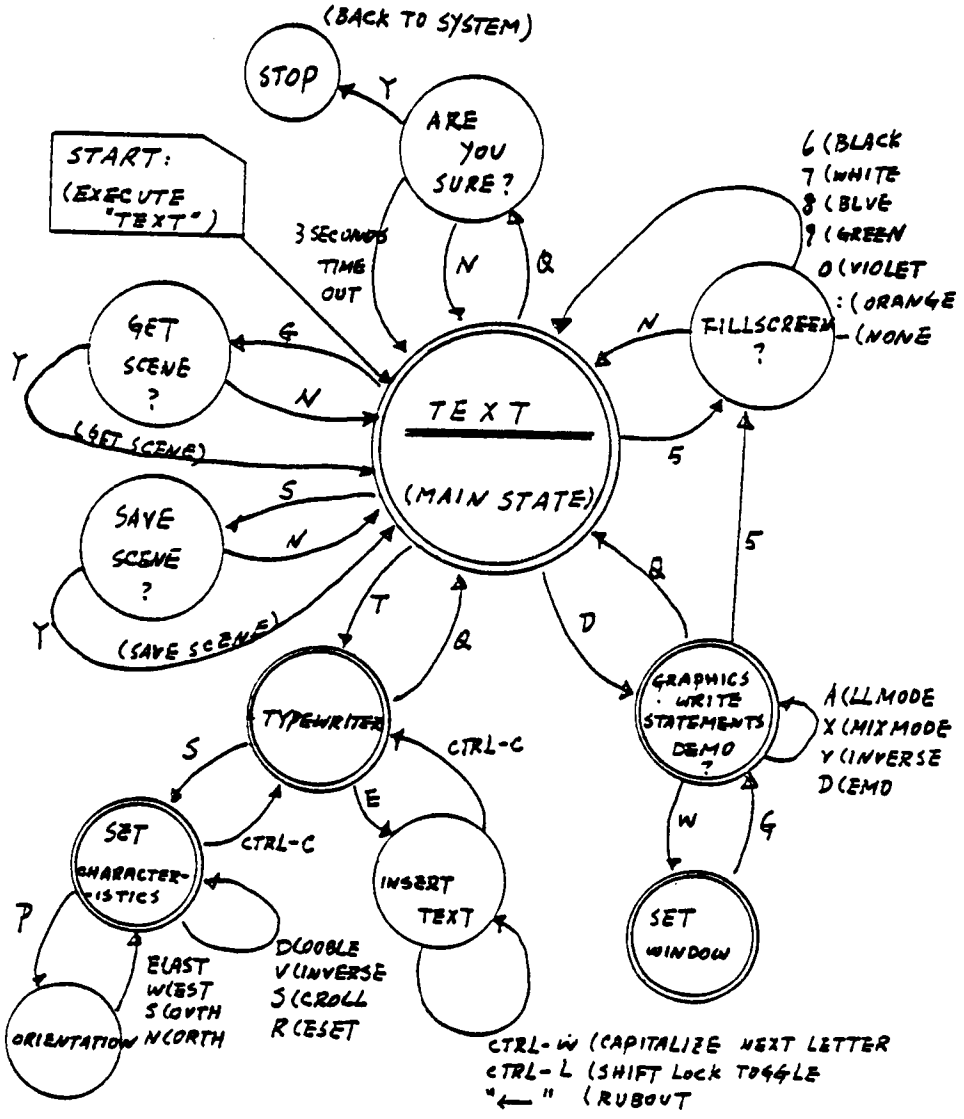


Figure 3-1. State Diagram for TEXT

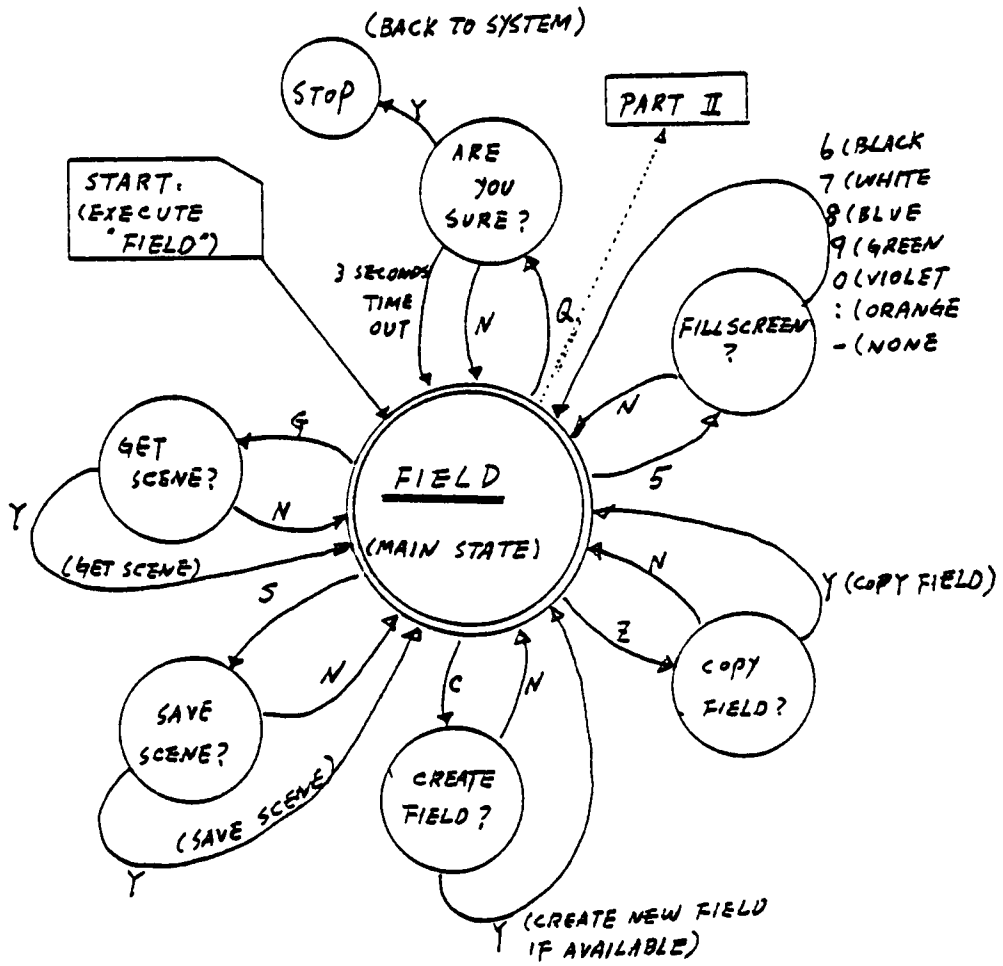


Figure 3-2. State Diagram for FIELD (PART I)

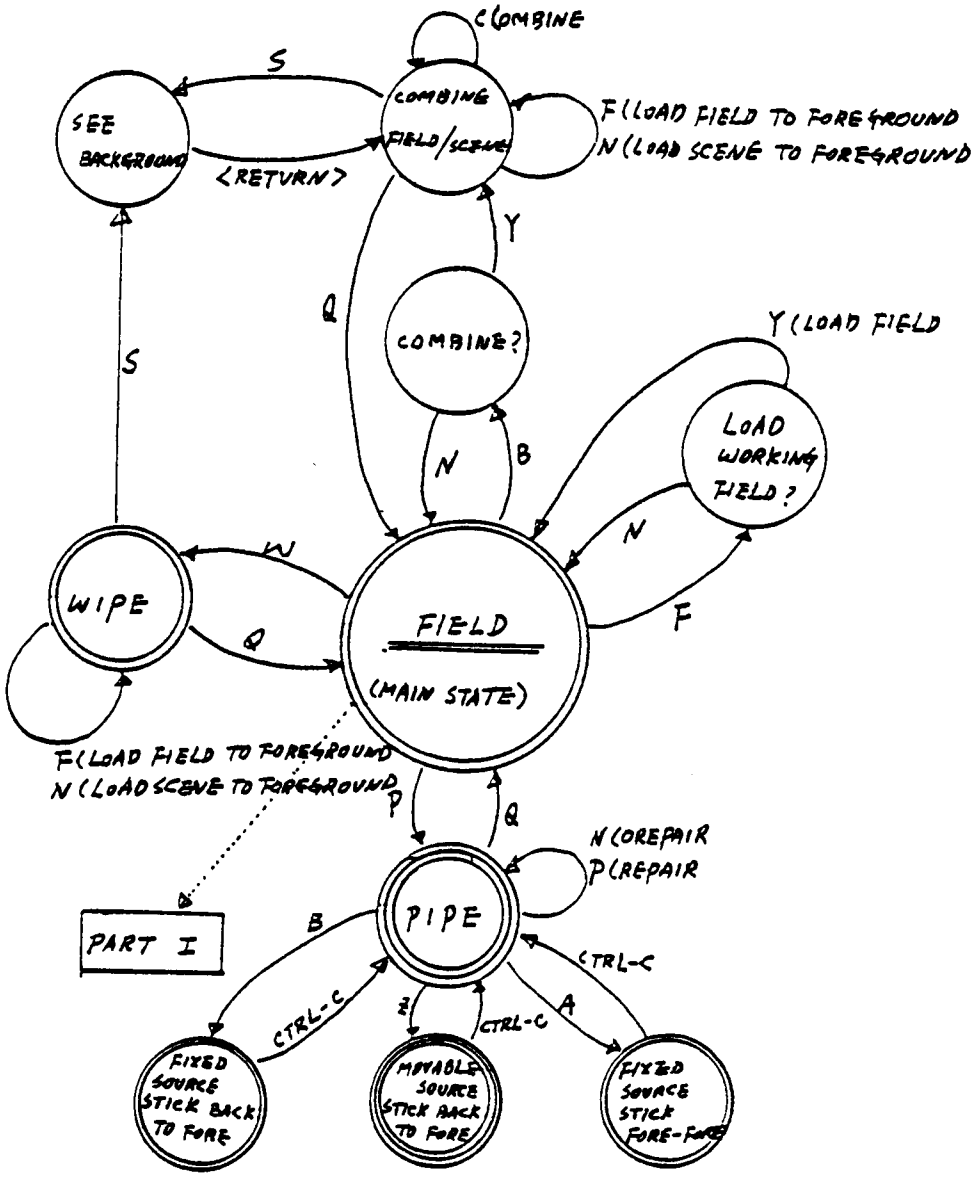


Figure 3-3. State Diagram for FIELD (PART II)

PADDLE 1. If Button(1) is pushed, the scene will be erased by moving the stick. The scale of the stick is set by the value of Paddle(1). The stick is also used as a diagonal to determine a rectangular text window (in the demonstration of the graphics write statements) or a solid image (in piping).

KEYPAD. The position of the stick on the screen can be controlled by a "keypad" (group of keys) around the key K (called the K keypad). Typing the I key causes the stick to accelerate upward one "pixel", the M key causes the stick to accelerate downward and to the right one pixel. The other keys (U J , L O) cause movement similar to their direction from K. In piping mode, another keypad around key D (called the D keypad) is used to control the destination stick (STICK2). As with the K keypad, the keys (W R S F X C) cause STICK2 movement in their direction from D.

CURSOR MOVEMENT. As soon as the TYPEWRITER function of TEXT is invoked by the user, a blinking cursor is displayed on the screen. The cursor is displayed as a horizontal or vertical line depending on the orientation case. The scale of the cursor is set to be single- or double-letter size. The cursor movement is controlled by typing a letter, the return key (the key marked return) or the rubout key (the key marked "<-"). The user does not need to type the return key when the text reaches the end of the screen edge, due to the feature of cursor wrap around. The return key only

enables the user to end the current text line without the requirement of text reaching to the screen edge. To move the cursor by using the return key does not erase any pre-existing letters or scenes in the text window. If scrolling is set to the EAST orientation case, the cursor stays at the bottom of the text window after it is moved there.

COLOR CONTROL. The color of the stick can be changed by typing one of the color key set. "6", "7", "8", "9", "0", ":", and "-" represent black, white, blue, green, violet, orange, and none respectively. The color of characters in TYPEWRITER or graphics write statements is also controlled by this color key set. However, in controlling the color characters, the none key (marked "-" on keyboard) is used to return colored letters to multicolor (not solid color) letters and return the stick to white color.

3.3 The TEXT Demo

Rasdazl uses programs TEXT and FIELD to show all the effects of WSTRLIB and FLDLIB. Here, we describe TEXT. The following section describes FIELD.

To make a scene with text strings, you should execute TEXT; it shows a main menu (Figure 3-4) at the bottom of the screen.

```
S(AVE    5(FILLSCREEN
G(ET     T(YPEWRITER
Q(UIT    D(GRAPHICS WRITE DEMO
```

Figure 3-4. Main Menu for TEXT

FILLSCREEN. To get a new screen color you can use this command. After typing 5, you are told to use one of the keys 6, 7, 8, 9, 0,:,- which stand for black, white, blue, green, violet, orange and none respectively. The screen shows the color you choose as soon as you hit any one of the above keys. Any other key causes no action. Note: this command will destroy any pre-existing scene on the screen.

GET. This is invoked when you want to get a scene from disk. After you type G, a question will be shown at the bottom of the screen to ask you for the name of a scene to be loaded. You should type the name and hit the RETURN key. It then loads the scene into both the foreground and background. If there is no such file on the disk, you will get an appropriate error message.

SAVE. When this command is invoked, it asks you for the name of the scene to be saved. It then writes the foreground scene out to the disk. If there was a previous scene on the disk with the same name, the new one replaces it.

GRAPHICS WRITE STATEMENT DEMO. When this command is invoked, a sub-menu is displayed on the bottom 4 lines of

the graphics area. (This sub-menu used SWRITELN, the graphics write line procedure. The text window defaults to the bottom 4 lines of graphics area). Figure 3-5 Shows this sub-menu.

```
5(FILLSCREEN   A(LLMODE     X(MIXMODE
D(EMO    Q(UIT  V(INVERSE    W(INDOW
```

Figure 3-5. Menu for DEMO

FILLSCREEN is the same as FILLSCREEN in the main menu. If ALLMODE is invoked, whole screen is used as a graphics area and the above sub-menu shows at the bottom of the screen. MIXMODE changes it back to mixed graphics mode. WINDOW is used to relocate the text window. After you type W, it asks you to set window position and to type G for "GO" when you get the window set. You position the stick so that it is the diagonal of an imaging box, the text window. The angle can be controlled, ranging from 0 to 90 degrees; the stick that is used to set the window size is colored orange. Immediately followed your typing G, the menu is then displaying in this new text window and all the messages stay in this window unless you set another text window again. You can use color key set ("6", "7", "8", "9", "0", "-") to change the color of characters. INVERSE changes the characters to be display in inverse mode. Up to now, all the messages you read were displayed on the graphics area because they used the graphics write statement called

SWRITELN. You can type D to display the effects of other graphics write statements, such as SWRITE, IWRITE, IWRITELN, SCWRITE. QUIT exits this mode and returns the control to the main state with mixed text/graphics mode.

TYPEWRITER. A menu for TYPEWRITER shown in Figure 3-6 comes up when T is pressed while in TEXT's main state. The characteristics of characters defaults to single-size, EAST, multicolor, non-scrolling, normal (non-inverse). To set other characteristics you should type S.

```
S(ET CHARACTERISTIC MODE
E(INSERT TEXT MODE
Q(UIT TYPEWRITER
```

Figure 3-6. Menu for TYPEWRITER

After you typing S, another menu is invoked. Figure 3-7 shows the menu. Then you can set the characteristics for the characters you type in insert text mode. In this mode, the color of characters is set by one key of the color key set.

```
D(OUBLE-SIZED    P(ORIENTATION
R(ESET    V(INVERSED-LETTER
S(CROLLNG CTRL-C:QUIT
```

Figure 3-7. Menu for Set Characteristics

RESET resets all characteristics back to the default mode. In the orientation command, the program asks you to type one

of the cases E(AST, W(EST, N(ORTH and S(OUTH. Any one of these keys returns you to the set characteristic mode. Except for ORIENTATION and CTRL-C, any command invokes a proper message to tell you what characteristic you just set and ask you to press RETURN to continue. A CTRL-C returns you to TYPEWRITER mode. The characteristics of characters will not change during the program execution until you set them again. Before you insert text by typing E, you should move the stick to the place where you want the text to be started. After E is pressed, another menu is displayed, which is shown on Figure 3-8, and a blinking cursor is shown on the screen.

```
CTRL-L(OCK KEY    "<-"(RUBOUT
CTRL-W(CAPITALIZE NEXT LETTER
CTRL-C(END OF INSERTION
```

Figure 3-8. Menu for Insert Text

Then every character you type will be at the place where the cursor is positioned. If you miss-type a letter, you should press the RUBOUT key (warked "<-") to erase this letter. If the letters you type are displayed as capitals, then you are in shift lock mode. This mode may be toggled (turned on and off) with a control-L. If you are not in the shift lock mode (letters you enter are displayed as lower case), you may get a single capital letter by preceding the letter with a control-W. A control-C ends the insertion and returns you to TYPEWRITER mode.

QUIT. If you want to quit the program, you should type Q. A question message is then invoked, such as "ARE YOU SURE? (Y/N)", if you want to quit the program, you should respond with "Y", else "N" returns you to the main menu. If you press no key for three sceonds after message, it assumes that the answer is no and returns you to the main menu.

3.4 The FIELD Demo

To make a scene and manipulate a solid image, you should execute FIELD. The effects of some of the commands in this program are the same as in TEXT. The following discussion describes only the different ones. Figure 3-9 shows the main menu of FIELD. At end of the main menu, there is a message to tell the user which field is currently displayed as the foreground and how many fields are available to be created.

```
P(IPE    S(AVE    B(COMBINE    Z(COPY
W(IPE    Q(UIT    G(ETSCENE    C(REATE
5(FILLSCREEN    F(LOAD WORKING FIELD
```

Figure 3-9. Main Menu for FIELD

In the following discussion we refer to two terms, fields and scenes. A field is a "scratchpad" image for use during editing, but a scene is a named disk file for permanent storage. This program maintains up to 4 fields, numbered 1..4.

COMBINE. When this command is invoked, it asks you "COMBINE FIELD/SCENE?". If you type "Y", it then invokes a menu like Figure 3-10.

```
F(LOAD EXISTING FIELD FROM DISK
N(LOAD EXISTING SCENE FROM DISK
S(EE BACKGROUND
C(OMBINE   Q(UIT
```

Figure 3-10. Menu for COMBINE

You should type F to load a pre-created field from disk into the background or N for an existing scene. Either you type F or N, it asks you to type a Field number or a scene name respectively. If you forget what scene or field you have loaded into the background, you can type S. The program displays the scene in the background. Pressing the return key returns you to viewing the foreground. To combine the background and foreground, you should type C.

CREATE. You are asked "CREATE NEW FIELD?", when you invoke this command. A return key returns you to main menu. If you respond with a "Y", it then writes the scene in the foreground to the disk and clears the foreground and background. Control then returns to the main state. The message in the main menu is updated and indicates what field you are working on now and how many fields are still available. If it indicates 0 available fields and you invoke this command again, you will get an error message such as "NO MORE FIELDS AVAILABLE!!". Then you should type RETURN to go back to normal operation.

COPY. If you want to copy one field to another field, you should type Z. Then you should enter the number of the source and destination fields according to the question message. If any one of the fields is not pre-created, you get an error message. After copying, nothing changes in the source field, but the scene in the destination field is replaced by the scene in the source field.

LOAD WORKING FIELD. If you want to switch the working field (the field on the foreground), you should type F. This command loads the field that you ask for into the foreground and saves the current working field on the disk. If the field you ask for has not been created, you get an error message such as "FIELD NOT CREATED/ILLEGAL FIELD!!!".

WIPE. This command invokes a menu and a warning message like Figure 3-11. The effects of LOAD FIELD, SCENE and SEE BACKGROUND are the same as those commands in COMBINE mode. Stick movement wipes the scene in the background into the foreground. That is, as the stick moves, pixels across which it passes are copied from the background to the foreground. It is best to set the stick on the proper area where you want to wipe before invoking this command. If you are already in WIPE mode, you can set the scale of the stick to be as small as possible (one pixel) with the PADDLE(1) control and then move the stick to the target area. Then, set the scale to wipe a scene with stick movements.

```
F(LOAD FIELD    S(EE BACKGROUND
N(LOAD SCENE    Q(UIT
!!WARNING: STICK MOTION WILL DAMAGE THE
  FRONT SCENE!!
```

Figure 3-11. Menu and Warning Message for WIPE

PIPE. This command is used to "drag" a region of a given field up, down, right or left. The PIPE menu is shown in Figure 3-12.

```
P(REPAIRED    N(OREPAIRED    Q(UIT
A(FORE-FORE    B(FIXED SOURCE BACK-FORE
Z(MOVABLE SOURCE BACK-FORE
```

Figure 3-12. Menu for PIPE

Two sticks are available and allowed to move in this command. The white stick is used as a source pointer and the orange as a destination pointer. PIPE will move a rectangular image, of which the stick is the diagonal. Stick movement is controlled by the K keypad and the D keypad. To pipe (drag) an "image" (a portion of scene), you should move the source stick on top of this image and set the scale and angle of the stick to cover this image. The scale and angle of sticks is controlled by two paddles; the angle you can control in this command is 0 to 90 degrees (The scale and angle of the destination stick are always slaved to the source stick). Then, you can type one of the following commands to pipe the source image to the place that the destination image is located.

1. FORE-FORE: Take source image from foreground to the destination image in foreground with fixed source image (source stick is not allowed to move).

2. FIXED SOURCE BACK-FORE: Take source image from background to destination image in foreground with fixed source image.

3. MOVABLE SOURCE BACK-FORE: Take source image from background to destination image in foreground with movable source image (source stick is allowed to move). PIPE always comes up with the REPAIRED mode (the scene in foreground is not damaged during piping). If you want to see the effects of NOREPAIRED, you should type N before entering into any one of the above modes.

CHAPTER IV

IMPLEMENTATION

4.1 SYSTEM.CHARSET

To manipulate character strings in the high-resolution graphics area is the most difficult task of this project. In order to display ASCII characters on the screen, there must be a set of computer coded characters available for this purpose. An UCSD standard character set file called SYSTEM.CHARSET is used in this project. There are 128 characters, a full character set, included in SYSTEM.CHARSET. Each character is coded as 8 bytes of 8 bits each and is displayed on the screen in an 8 high by 7 wide grid, the high order bit is not used. SYSTEM.CHARSET needs 1,024 bytes of memory space and can be loaded into any location of the RAM (Random Access Memory) area. In this project, a special area of the computer RAM is used for SYSTEM.CHARSET. This area, starting at location 1800 (hexadecimal), is used as a utility buffer by the Rascal system. Figure 4-1 illustrates the memory-map of SYSTEM.CHARSET. Each character corresponds to a specific sequence of eight bytes in this memory. The program fetches these eight contiguous bytes and puts them on the screen as the eight rows of a character. SYSTEM.CHARSET is loaded

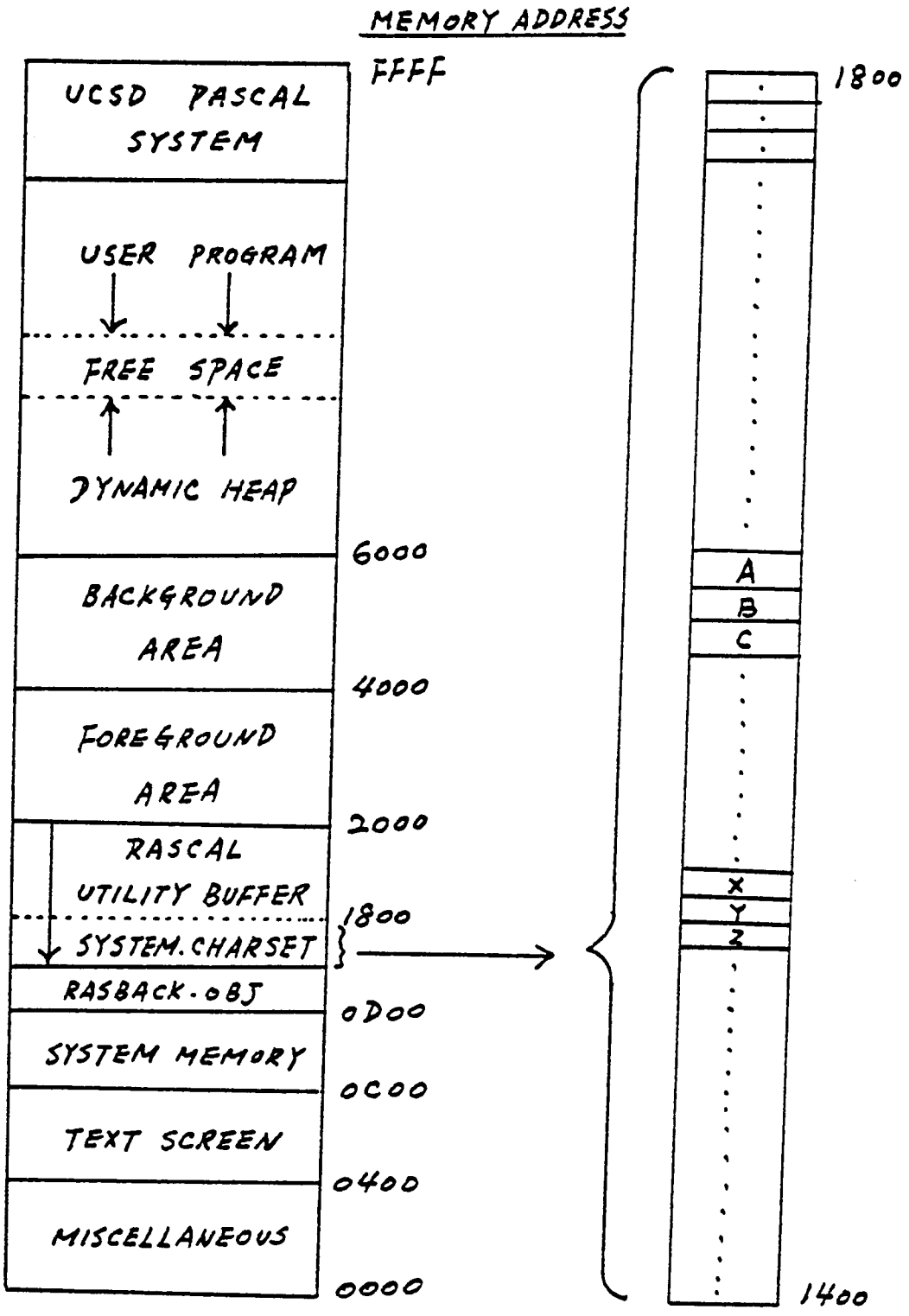


Figure 4-1. Memory-map for SYSTEM.CHARSET

from disk whenever the program needs to use the character set. There are two procedures called WCHAR and WSTRING which use SYSTEM.CHARSET whenever they are activated by the program. Once SYSTEM.CHARSET is loaded into the memory, it will reside in the memory unless this memory is disturbed by other uses of the Rascal utility buffer. Therefore, there is no reason that SYSTEM.CHARSET should be loaded into memory every time when WCHAR or WSTRING is called. Rasdazl uses one procedure called LOADSYS to check if the loading is necessary. In LOADSYS, two check points are examined. If these two values are not equal to the values that should be at those locations in SYSTEM.CHARSET, then LOADSYS loads the SYSTEM.CHARSET into memory.

4.2 Oriented and Double-sized Characters

In SYSTEM.CHARSET, each character consists of 8 bytes and each byte is a unique 7-bit value called an ASCII byte. To display a character on the screen, the program pokes (stores) this ASCII byte in the high-resolution memory and the screen displays this byte as a line of tiny dots (pixels). Therefore, to re-orient a character requires dealing with a matrix transformation only, if we take each character as an 8 by 8 matrix (called the character matrix). There are four kinds of orientations in Rasdazl; they are EAST, WEST, NORTH and SOUTH orientations. These terms are used on the basis of the direction that a text goes. Clearly, the EAST orientation is the normal left-to-right

direction. In the Apple high-resolution graphics memory, each dot on the screen represents one bit and seven of eight bits in each byte are displayed on the screen, with the remaining bit used to select the colors of the dot in that byte. This select bit is called the undisplayed bit (or phase bit). From the user's standpoint this undisplayed bit is the most significant (eighth) bit in a byte. However, from the system's standpoint, it is the rightmost bit. The reason for this is that every byte that is poked into the high-resolution memory is swapped by the system hardware. Figure 4-2 shows this porcess. Discussion will be based on the screen image.

In the WEST orientation, every bit is reversed in an ASCII byte when this byte is acquired from the SYSTEM.CHARSET. To take care of the undisplayed bit, every byte is shifted one bit left. Figure 4-3 illustrates this transformation.

In the NORTH and SOUTH orientations, the columns and rows are swapped in each character matrix. Figure 4-4 shows the transformed character E in these two different orientations. Again, every byte needs to be shifted left one bit for the SOUTH orientation to prevent it from losing the least significant bit. One more problem arises in the NORTH and SOUTH orientations. The density of upper-case characters in SYSTEM.CHARSET is 6 x 7 (width by height)

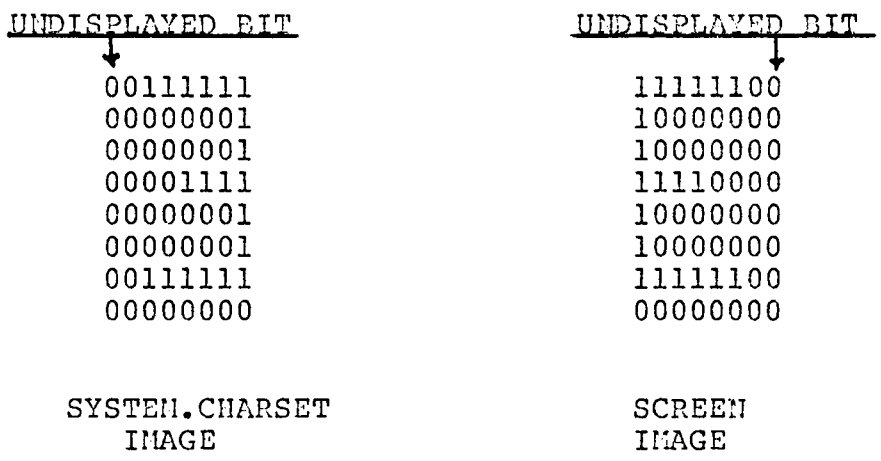


Figure 4-2. Character E in SYSTEM.CHARSET and Its Screen Image

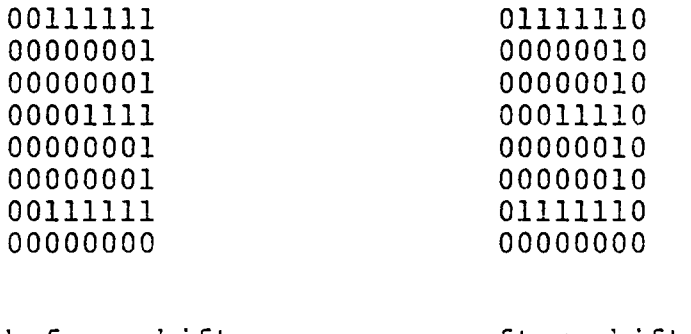


Figure 4-3. Character E after Transformation

Figure 4-4. Transformation of Character E in the NORTH and SOUTH Orientations

(lower case ranges from 5 x 5 to 5 x 7) and only 7 bits of each byte are displayed on the screen. After transformation, the density of the NORTH and SOUTH characters has been changed to 7 x 6. Thus, there will be no space bit between two adjacent characters; see Figure 4-5. Therefore, the characters will be ambiguous and sometimes unreadable for certain adjacent pairs of characters on the screen. To overcome this problem, one additional byte is skipped between characters.

For double-sized characters, a "doubling" operation is performed. Every bit in each ASCII byte is doubled, and every byte is poked into the high-resolution memory twice. Figure 4-6 shows the double-sized character E. The logic of rotating double-sized characters is identical to the rotation of single-sized characters. Transformation is performed before the doubling operation if transformation is needed.

4.3 Solid Color Characters

The characters displayed on the screen that we have discussed so far are all multicolor characters. Multicolor means each character appearing on the screen is not a unique color. It could be a white color mixed with green ,violet, orange or blue. Solid (unique) color characters are very difficult to handle due to the special characteristics of color construction in the Apple II microcomputer. There are six colors (black, white, green, orange, violet and blue)

```
000000000000000
000000000000000
100000110000001
100000110000001
100100110010001
100100110010001
100100110010001
111111111111111
```

Figure 4-5. Adjacent Characters E in the North Orientation

```
1111111111110000
1111111111110000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1111111100000000
1111111100000000
1100000000000000
1100000000000000
1100000000000000
1100000000000000
1111111111110000
1111111111110000
0000000000000000
0000000000000000
```

Figure 4-6. Double-sized Character E

available in the Aple high-resolution graphics mode, subject to the following limitations:

1. If a bit in the high-resolution graphics memory area (picture buffes) is "off", its corresponding dot will be black.

2. Dots in even columns will be violet or blue depending on whether the undisplayed bit is set or not.

3. Dots in odd columns will be green or orange depending on whether the undisplayed bit is set or not.

4. Two color dots side by side always appear white, even if they are in different bytes.

A color table (which is shown in Figure 4-7) is then set accordingly to make solid color characters.

From the color table, it is easy to see that the color represented by 10 or 01 is reversed in odd and even bytes. To produce solid characters, we are not so lucky that every "ON" ("1") bit in every ASCII byte falls into the right position, without damaging the characters' shape. Therefore a special process is used to overcome this problem. 6 bits are used from each ASCII byte. Besides the undisplayed bit, there is one bit left to be used as a space bit (to separate two adjacent characters). In making solid color characters, this space bit is sacrificed. Every ASCII byte is shifted one bit and "OR"ed with its original byte, then "AND"ed with the correct color byte in the color mask table. The sliding and "OR"ing process assure that ON bits occur in pairs; so one of them will survive being "AND"ed with a color mask

byte. The above logic only applies to the EAST and WEST orientations. For the NORTH and SOUTH orientations, every character matrix has been transformed to a matrix with density 7 x 6. Therefore it is very difficult (in fact it seems impossible) to find a method to implement solid colors for the NORTH and SOUTH orientations with the standard SYSTEM.CHARSET and without damaging the character shape. That's why solid-color, single-sized characters in the NORTH and SOUTH case are not presented in this project.

To color double-sized characters, the logic is similar to single-size characters. There is a special problem with characters having a vertical center line (T, I). This is because some characters have ON bits which happen to fall into the adjacent pair of 0's in the odd and even masks for a given color. For instance, Figure 4-8 shows the double-sized character "T" and the color masks for solid green. If we did not perform a "shift-or" logical operation and just used the AND operation for each ASCII byte in a double-sized character T, an awkward result would be obtained.

4.4 Application of WCHAR and WSTRING

WSTRING is a procedure to write a text string in the high-resolution graphics area with a given X and Y position, while WCHAR only writes a character in the graphics area. The demonstration of WCHAR is called a "terminal-like typewriter" in this project. The reason that it is called this is that it has some features commonly found on CRT

COLOR	ODD BYTE	HEX	EVEN BYTE	HEX
BLACK	00000000	00	00000000	00
WHITE	01111111	7F	01111111	7F
GREEN	00101010	2A	01010101	55
ORANGE	10101010	AA	11010101	D5
VIOLET	01010101	55	00101010	2A
BLUE	11010101	D5	10101010	AA

Figure 4-7. Color Table

```
0000111111111100
0000111111111100
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000011000000
0000000000000000
0000000000000000

0101010100101010 GREEN PAIR OF MASKS
```

Figure 4-8. Double-sized T and Green Pair of Masks

terminals. These included cursor positioning of text, a rubout key, and scrolling characteristics (scrolling is only applied to EAST orientation).

There are several rules (assumptions) that are followed by the demonstration program:

1. A blinking stick (blinking with green and black) is used as a cursor and its size tells single- or double-sized characters.

2. A text window is determined by the current position of the stick and the screen edges (Figure 4-9 shows four kinds of text windows).

3. Text entry starts from the current position of the stick on the screen.

4. When the cursor reaches the edge of the text window (one side of the screen), it wraps around (horizontally) as if a carriage return were hit.

5. When the cursor runs out the bottom of the text window, it wraps around (vertically) and starts from its original starting position. For the EAST case, if scrolling is set by the user, the text will be scrolled upward instead of wrapping around.

6. The RETURN key on the Apple keyboard works like a carriage return in text mode.

7. The "<-" (back arrow) key is used as a rubout key. If we try to rubout across the left margin of the text window, it wraps around (horizontally) backward, to the right end of that line.

8. The rubout is implemented by painting the deleted character with the current screen color.

9. The rubout function only works within the text window, i.e., the cursor stays at the text starting location if you try to delete the character beyond this position.

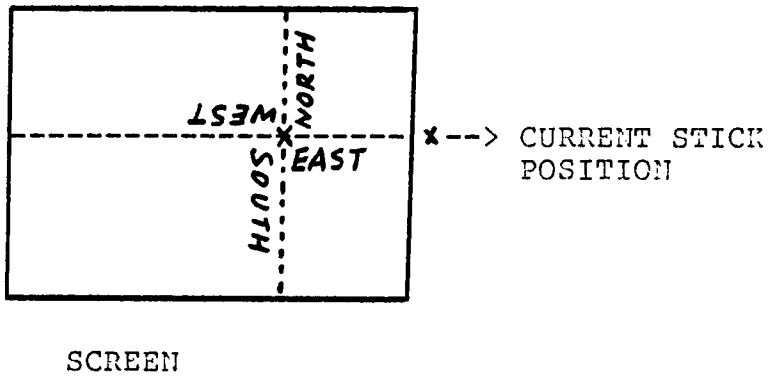


Figure 4-9. Four Kinds of Text Windows

The most difficult task in this demonstration is to handle the cursor movement and text scrolling. Initially, the cursor is set to be a character size (single or double) and is displayed as a horizontal line for the EAST and WEST cases or vertical line for the NORTH and SOUTH cases. These steps are done by a procedure called SETCURSOR. Once a character is typed, WCHAR is called to write a character to the place that the cursor is positioned on the screen. Then the cursor moves to the next character location. If the rubout key is typed, the cursor moves backward and a procedure RUBOUT is called to erase the character indicated by the cursor.

The starting position of a text on the screen is not guaranteed to be a location of which X is power of 7 and Y is power of 8. This situation should be taken into consideration in arranging the cursor wrapping around. In other words, if the cursor is moving close to the window edge, there must be some way to determine that the remaining room on the screen is big enough for the next character. If it is, then a regular movement occurs. Otherwise, wrap-around occurs. Two module functions called XYMOD1 and XYMOD2 are used to handle this job. These functions return two values; one is where next character should be located (either wrapping around or not), and the other is the flag value to tell if the remaining room is big enough or not.

Scrolling is performed by an assembly routine called SCROLL. The secret to understanding how the scrolling works is to realize that SCROLL treats a text string as 8 single lines on the screen and moves the rest of the lines (starting at the ninth line from the top of the text window) up 8 lines (one character line). After the line movement, SCROLL then fills the bottom character line in the window with the current screen color. Figure 4-10 shows a character line on the screen.

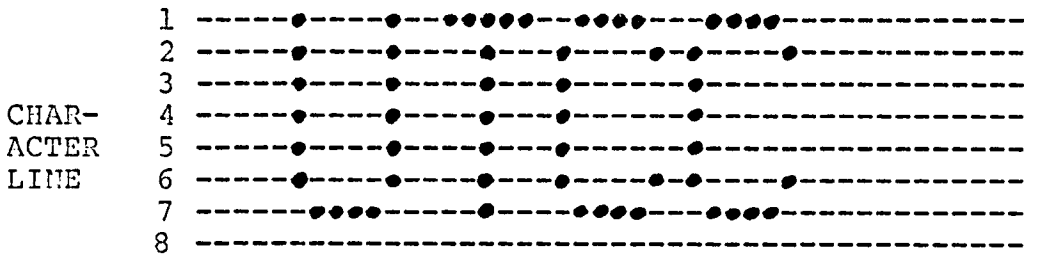


Figure 4-10. 8 Lines Represent One Character Line on the Screen

The demonstration of WSTRING is a set of graphics write statements. The format of these write statements graphics area. The format of these write statements and their equivalent Pascal write statements are shown in Figure 4-11.

```
SWRITE(STROUT);          WRITE(STROUT);
SCWRITE(CHAROUT);        WRITE(CHAROUT);
SWRITELN('');            WRITELN;
SWRITELN(STROUT);        WRITELN(STROUT);
IWRITE(IVAL);            WRITE(IVAL);
IWRITELN(IVAL);          WRITELN(IVAL);
```

Figure 4-11. Graphics and Equivalent Pascal Write Statements

STROUT, CHAROUT, and IVAL are variables that are declared as STRING, CHAR, and INTEGER respectively. To set the letter characteristics (color, orientation, etc) for these write statements, a procedure called SETCHD is accessible to the user.

4.5 Field Manipulation

In the Rascal system, there are two 8-K byte picture buffers called foreground and background. To enhance the

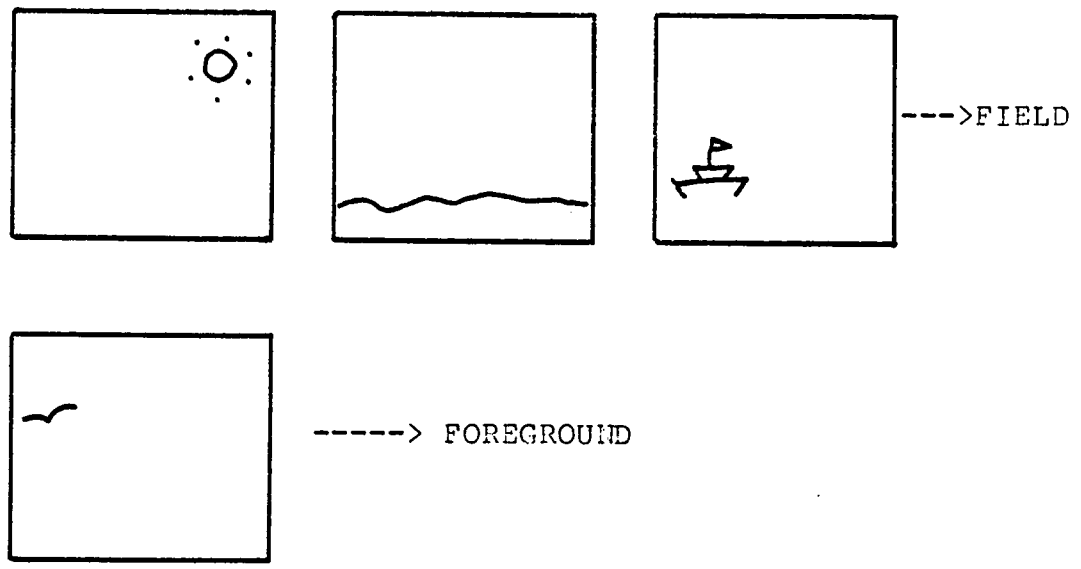
picture editing, three extra picture buffers may be created by the user. These three picture buffers along with the foreground are called "fields" in this project. Of cause, the background is another field; however, it is used as a temporary buffer in most of the time, and is often destroyed. Due to memory size limitations, these three extra fields reside on the disk which is assigned to volume 5 (disk drive 2). To handle all disk access in field manipulation, a procedure called FILEIO is used. Its functions include creation of a new field, and getting or saving an existing field. If there is any I/O error, FILEIO also invokes an appropriate error message.

Fields can be created when the user issues an appropriate command and are accessible to the user later on, as needed. A procedure SWITCHF is used to switch fields (one on the disk and one on the foreground). SWITCHF performs the following steps when it is called.

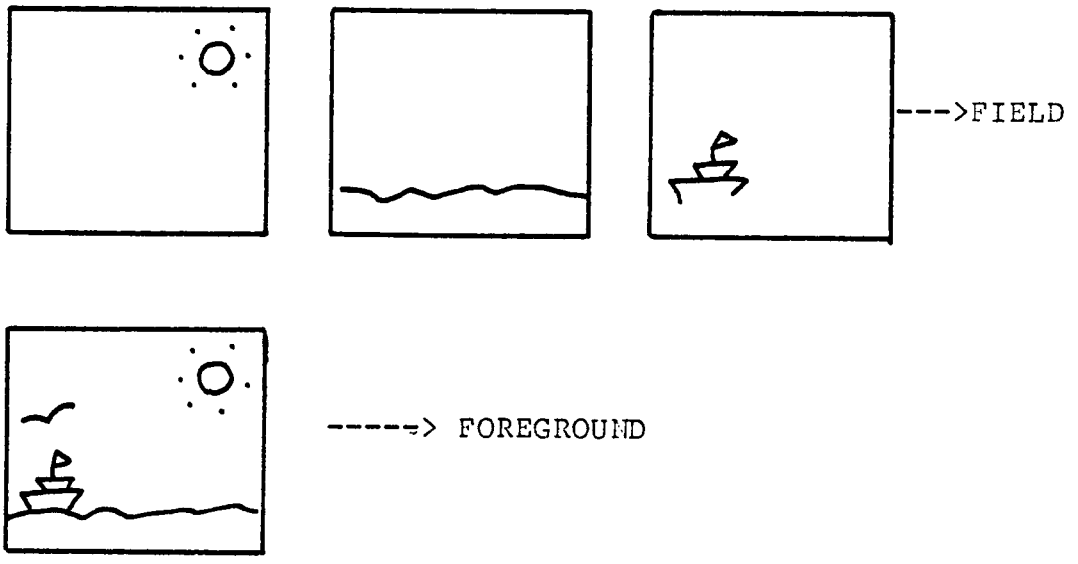
1. Load desired field into background
2. Write foreground to the disk, replacing old disk fild
3. Copy background into foreground

One of the important features of field manipulation is field combination. Field combination is performing an "OR" operation for every byte in the foreground scene with the corresponding byte in the background scene. All these jobs are done by an assembly routine COMBF. Figure 4-12 shows the effects of four field combinations.

Rasdazl also provides a convenient feature for picture editing called "wiping". There is a stick that is available to the user in the FIELD program. The stick is a movable figure and can be controlled by K-keypad. Most of the time in normal animation the foreground scene is identical to the background scene. During figure movement, the corresponding pixels are copied from the background scene into the foreground scene to repair the damage as the figure moves. In wiping mode, the user can load a field or a scene from the disk into the background only and move the stick to "wipe" a region of the scene in the background into the forground. The size of a region is determined by the scale of the stick and the direction that the stick is moving. Figure 4-13 illustrates the wiping operation. Since stick movement damages the foreground scene in this mode, a warning message is always shown in the wiping menu.

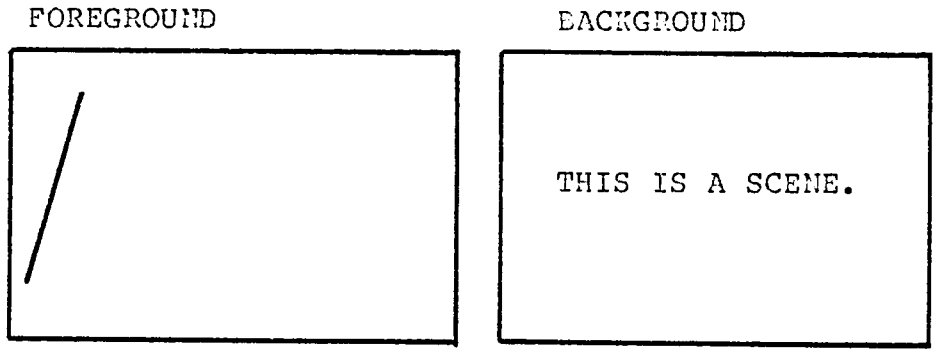


(A) BEFORE COMBINATION

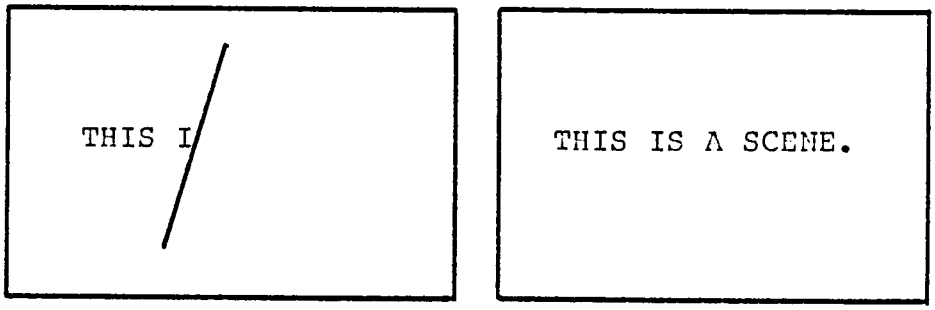


(B) AFTER COMBINATION

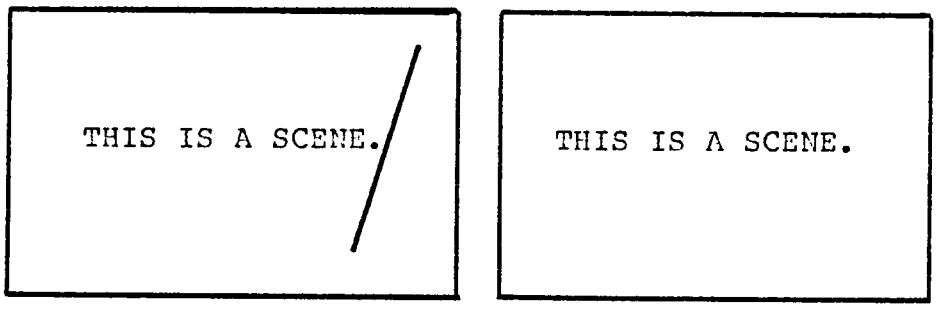
Figure 4-12. The Effect of Fields Combination



(A) BEGORE WIPING



(B) IN THE MIDDLE OF WIPING



(C) AFTER WIPING

Figure 4-13. Wiping Operation

4.6 Solid Image

A solid image is a rectangular region of a given field or scene that can be indicated by a stick (the diagonal of the rectangle). Rasdazl uses a data structure shown in Figure 4-14. to implement the solid image manipulation.

```
SOLIDFIG = RECORD
             XLOC:INTEGER; (* X POSITION OF SOLID IMAGE *)
             YLOC:INTEGER; (* Y POSITION OF SOLID IMAGE *)
             XLEN:INTEGER; (* WIDTH OF SOLID IMAGE *)
             YLEN:INTEGER; (* HEIGHT OF SOLID IMAGE *)
           END;
```

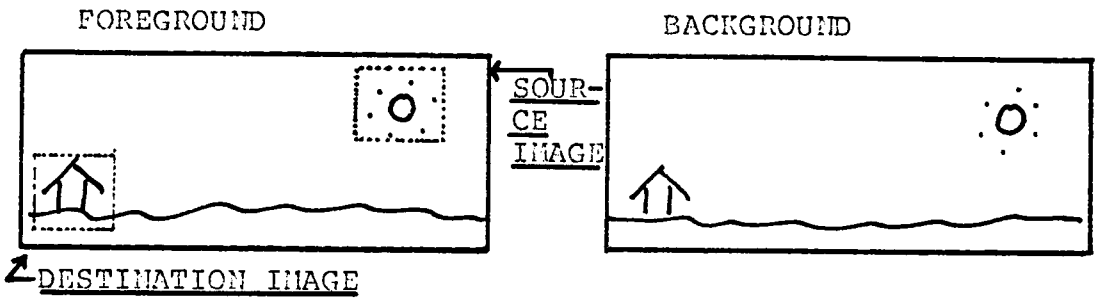
Figure 4-14. Data Structure for Solid Image

The XLOC and YLOC form a coordinate pair to give the location of the solid image on the screen. These values are obtained from two Rascal functions WHEREX and WHEREY, applied to the figure stick. As long as the stick is turned on (by using Rascal procedure TURNON), WHEREX and WHEREY gives the current stick position on the screen. The XLEN and YLEN give the size of a solid image; they are obtained from Rascal procedure FAREND. FAREND gives a coordinate pair (X,Y) indicating where the far end of the stick is.

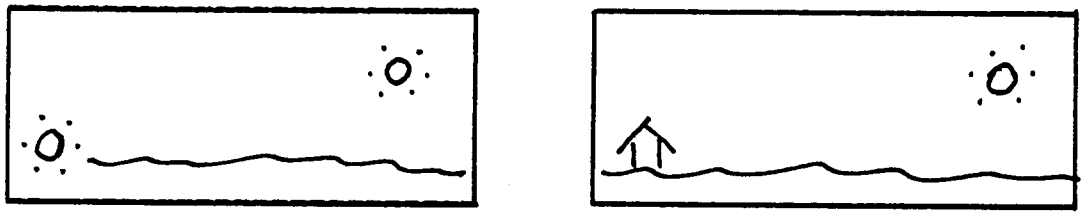
A solid image is then defined by the current position and the scale of a stick. Rasdazl uses two sticks to define source and destination solid images. Some of the information about STICK2 (XLEN, YLEN, angle and scale) is slaved to STICK, but its position is controlled by the D-keypad. The angle of both sticks can be controlled by Paddle(0) between 0 to 90 degrees.

To pipe a solid image, Rasdazl calculates the location of source and destination sticks on the screen and copies every byte in the source image into the destination image with the given information (XLEN AND YLEN). All these jobs are done by an assembly routine called PIPEACT. Rasdazl also provides the user with a choice, to repair or not to repair the scene on the foreground while piping. If the user chooses the mode that is repairing, then another assembly routine is used, called WIPEOUT. WIPEOUT wipes a given region from the background scene into its corresponding foreground scene. While a solid image is moving, Rasdazl calls WIPEOUT to repair its old images. Figure 4-15 shows piping with repair requested. The steps of solid image manipulation are summarized as follows:

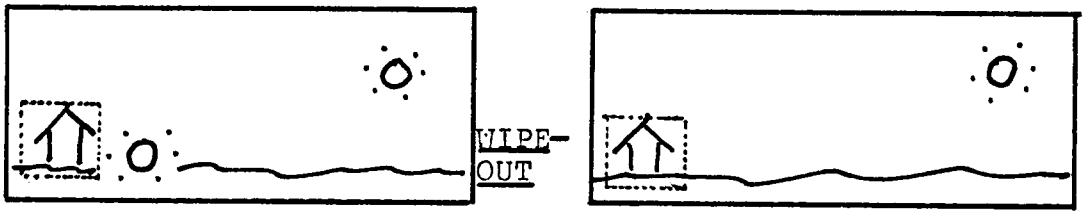
1. Turn on STICK and STICK2
2. Slave STICK2 to STICK
3. Get information for source and destination images
4. If movable source is required, get information for source image
5. If repair is requested, call WIPEOUT
6. Get new information for destination image (might be the old information if the destination image was not moved by the user)
7. Call PIPEACT
8. Go to step 4 until Control-C is hit by the user
9. Copy foreground scene to background scene



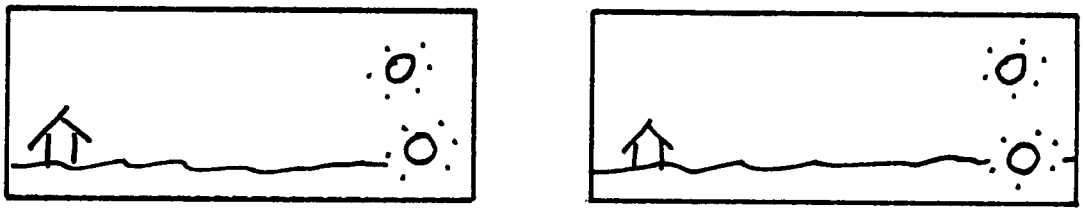
(A) BEFORE PIPING



(B) PIPING START



(C) IN THE MIDDLE OF PIPING



(D) AFTER PIPING

Figure 4-15. Piping Effects with Repair Requested

The logic describes above is pretty straightforward and seems all right at first glance. However, an awkward result was obtained in testing the above logic. PIPEACT and WIPEACT calculate the high-resolution memory with a given pair of XLOC and YLOC, then perform copy operation with a given pair of XLEN and YLEN. There is nothing wrong in the memory calculation. But if the solid image is piped across the screen edge, then the image is cut into pieces and spread on the foreground scene. This is overcome by adding an extra step in step 7. This extra step can be discussed more clearly by using the diagram in Figure 4-16.

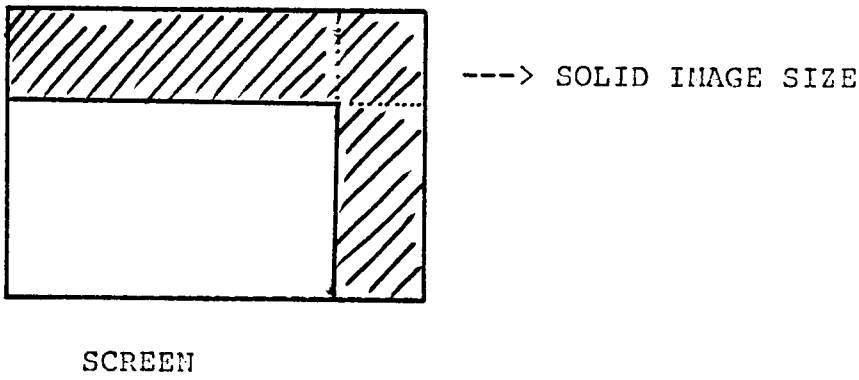


Figure 4-16. Diagram of Danger Region for Piping

The shaded region is determined by XLEN and YLEN of a solid image. As soon as the entire stick touches the top or side of this shaded region, step 7 is skipped (not to get new information for the destination image). If the user selects a movable source mode, then this test is applied in step 5 also.

CHAPTER V

CONCLUSION

5.1 Accomplishments

Speed is one of the key factors in evaluating an animation system. For this reason, several time-critical features of this project (the typewriter, text scrolling, and piping) were implemented in assembly language and this made the work more successful.

The typewriter allows the user to manipulate text strings in Apple II high-resolution graphics quite freely with the rubout feature and its blinking cursor makes the text insertion quite convenient. The Rascal system uses the mixed text/graphics mode that allows a text scrolling window at the bottom of the screen. The Graphics write statements remove this restriction. By using the graphics write statements, the text window can be relocated to any desired place, such as a side window.

Field manipulation is very convenient for picture creating and editing. The user can use up to four picture buffers and combine pictures together as many times as he wishes. In wiping mode, the user can wipe a selected region of an existing picture into another picture. In addition, a region of a given field or scene can be relocated by using the piping mode. One of the piping features (such as

movable source and destination sticks) can also be used to create a nice picture for artistic purposes.

5.2 Problems

There are a couple of remaining problems due to system limitations.

1. The quality of solid color letters is not as perfect as we had hoped.

2. Solid color is not available for single-sized letters in NORTH and SOUTH orientations

These problems are very difficult to correct with the standard SYSTEM.CHARSET. However, our chances will improve if we can create a new system character set with consideration given to the color organization of the Apple II computer. Actually, three different kinds of packages are available for such a purpose. The first one is Rascal MAKER which is a figure creator. The user can use MAKER to create different kinds of character sets. Similar work has been done by Kurt Kissell in August 1980. In his study with Dr. Michael Moshell of The University of Tennessee, he used MAKER to create a set of twenty-six Rascal figures A..Z that can be arbitrarily rotated, recolored, rescaled, and painted into the background. In his work, the character set used more space by far and is not as neat as Rasdazl.

The second one is the Keyboard Filter from Mountain Hardware Inc. The Keyboard Filter, a 2 K ROM chip, provides a program named "the Font Editor" that allows the user to

define his own character set. The keyboard Filter has functions similar to those of Rasdazl. Its creator faced the same problem of colored characters, even though he produced his own character set. Here is his note.[5]

> NOTE: The less-than-wonderful quality of the colored letters is a direct and unavoidable result of limitations inherent in the Apple II's high-resolution graphics. Even so, they are usually quite readable, and will generally look better if double spaced.

The last one is the character CREATOR program of Apple Pilot from Apple computer Inc. This package probably is the appropriate one that for us to use to design a special character set like the one that Atari uses to produce the colored characters.

5.3 Future Work

Several work are being considered in the near future to enhance the power of this project. They are:

1. Using Apple Pilot to create a special character set
2. Adding real number in graphics write statements set
3. Getting or saving a solid image from/on disk
4. Recoloring, rotating, and rescaling the solid image

The other future work is to implement a procedure that can "compress" all 4 fields into one video screen (another field). However, the possibility of this work is still in question.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Gentleware Corp. Rascal and Interpas Reference Manual, Knoxville, Tennessee, 1981.

2. Amann, G, W. MAKER: FIGURE CREATOR FOR RASCAL ANIMATION SYSTEM, M.S. Thesis, The University of Tennessee, Knoxville, 1980.

3. Mitchell Waite. Computer Graphics primer, Howard W. Sams & Co., Inc., 1980.

4. Atari, Inc., Atari Personal Computer System Operating System User's Manual, 1980.

5. Barney Stone. Keyboard Filter Software Manual, Mountain Hardware, Inc. 1979.

6. Apple Computer Inc., Apple II Reference Manual, 1979.

7. Apple Computer Inc., Apple Pascal Reference Manual, 1979.

8. Gaining market share is IBM's challenge, Mini-Micro System, October, 1981.

9. Phil Lemmons, The IBM personal Computer: first impressions, Bute-The Small Systems Journal, October, 1981.

10. Chris Crawford, The Atari Tutorial Part I, Byte-The Small Systems Journal, September, 1981.

11. Chris Crawford, The Atari Tutorial Part II, Byte-The Small Systems Journal, October, 1981.

12. Stan Miastkowski, Extended Color BASIC for the TRS-80, Byte-The Small Systems Journal, may 1981.

13. John Martellaw, Apple II HIRES Graphics Memory Mapping, Kilobaud-Microcomputing, September, 1980.

14. Tim Ahrens, Jack Browne, and Hunter Scales, What's inside Radio Shack Color Computer?, Byte-The Small Systems Journal, March, 1981.

15. Gregg Williams, The Commodore VIC20 Microcomputer: A low-cost, high-performance Consumer Computer, Byte-The Small Systems Journal, May, 1981.

APPENDIXES

APPENDEX A

FREEKEY

RASCALIN	setup routine for all Rasdazl programs
DISPLAY1	displays foreground on the screen
DISPLAY2	displays background on the screen
READKEY	interrupts stick control and returns an accepted key
COPYBACK	copies foreground into background

APPENDIX B

WSTRLIB

SETCMD	sets characteristics for graphcis characters
RESETCMD	resets characteristics of graphics characters to the default mode
BOTTOMSET	initializes the text window for graphics write statements
GETWINDOW	gets a new text window for graphics write statements
CGOTOXY	sends graphcis write statements to (X, Y) coordinates
WCHAR	writes a character to the graphcis area with a given location
WSTRING	writes a string to the graphics area with a given location
SWRITE	writes a string to a text window
SWRITELN	writes a string to a text window, ending the current line
SCWRITE	writes a character to a text window
IWRITE	writes an integer value to a text window
IWRITELN	writes an integer value to a text window, ending the current line
CWRITELN	a switch of Pascal WRITELN and SWRITELN

APPENDIX C

FLDLIB

FILEIO	reads, writes, and opens a disk file
GETFLD	gets a given scene into the FOREGROUND, BACKGROUND or BFGROUND
SAVEFLD	writes foreground scene to the disk
SWITCHF	swaps the foreground scene and a disk scene
COMBF	combines foreground scene and background scene
SETSFIG	gets data value for source image
SETDFIG	gets data value for destination image
WIPEOUT	wipes a region of background scene into the corresponding foreground location
PIPEACT	copies a source region either in foreground or in background into the place where the destination image is located

VITA

Jack Chung-Chih Wang was born in Taipei, Taiwan, Republic of China on February, 1951. He received his Bachelor of Science degree in Economics from The University of Chinese Culture in June 1974. After graduation, he worked with the First Commercial Bank in Taipei for four years.

In September 1978, he came to The University of Tennessee as a graduate student and began study toward a Master's degree in Computer Science. In January 1981, he was employed by Environmental Systems Corporation in Knoxville while he began to work on his thesis.

He received the Master of Science degree with a major in Computer Science in December 1981 and continues his career with Environmental Systems Corporation.