ATARI® PROGRAM EXCHANGE

Joel Gluck

fun-FORTH

Sound and graphics commands for use with EXTENDED fig-FORTH

Diskette: 24K (APX-20146)

User-Written Software for ATARI Home Computers

Joel Gluck

fun-FORTH

Sound and graphics commands for use with EXTENDED fig-FORTH

Diskette: 24K (APX-20146)

fun-FORTH

by

Joel Gluck

Distributed By

The ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

To request an APX Product Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/727-5603

TABLE OF CONTENTS

INTRODUCTION

OVERVIEW

Fun-FORTH is a special collection of FORTH words (commands) for use with EXTENDED fig-FORTH (APX-20029) by Patrick L. Mullarky. The programming (especially game programming) tools in fun-FORTH include words facilitating advanced sound generation, simple turtle graphics, use of the Joystick and Paddle Controllers, and use of the console keys and the keyboard. It also contains words supporting timekeeping, random number generation, creation of arrays, recursion, and other useful functions.

Fun-FORTH also contains QuadraSketch, a small drawing program demonstrating use of various fun-FORTH words.

REQUIRED ACCESSORIES

24K RAM
ATARI 810 Disk Drive
EXTENDED fig-FORTH (APX-20029)

OPTIONAL ACCESSORIES

One ATARI Joystick Controller
One set of ATARI Paddle Controllers

CONTACTING THE AUTHOR

Users wishing to contact the author about fun-FORTH may write to him at:

```
1096 George Court
Merrick, NY 11566
```

GETTING STARTED

LOADING fun-FORTH

1. Remove any cartridge from the cartridge slot of your computer.

2. Turn on your disk drive. When the BUSY light goes out, place the EXTENDED fig-FORTH diskette in the drive and turn on your computer.

3. The program will load into memory and the prompt "fig-FORTH 1.1" will display when the load is complete. Press the RETURN key to display the standard FORTH prompt "ok".

4. Now you may load any of the following packages from diskette. Use the appropriate number before the LOAD command and press RETURN after typing each command.

```
39 LOAD   (ASSEMBLER)
27 LOAD   (EDITOR)
50 LOAD   (GRAPHICS)
21 LOAD   (DEBUG)
81 LOAD   (OPERATING SYS.)
60 LOAD   (FLOATING-POINT)
```

5. To use fun-FORTH, load the ASSEMBLER and GRAPHICS packages. If you plan to program, the EDITOR is necessary and the DEBUG package is very useful.

   Notes.

   a. In loading the packages, ignore any "isn't unique" messages. These are supposed to appear.

   b. Once you start loading screens, don't press the SYSTEM RESET button. It will erase what you've loaded.

6. When you finish loading in the packages you want from EXTENDED fig-FORTH, remove the diskette and insert the fun-FORTH diskette.

7. Type

```
50 LOAD
```

and press RETURN. An introductory message will appear, followed by a request to type I to display an index of the names of all the screens available in fun-FORTH, or type anything else to exit fun-FORTH. (To display the screen index at other times, type 1 42 INDEX .) The screen index looks as follows:

```
0
1 ( fun-FORTH by Joel Gluck 1982 )
2 ( Joystick Words      fun-FORTH )
3 ( Paddle Words        fun-FORTH )
4 ( Keyboard Words      fun-FORTH )
5 ( Console Words       fun-FORTH )
6 ( Time Words          fun-FORTH )
7 ( UNUSED screens 7-13 )
8
```

```
 9
10
11
12
13
14 ( ERROR MESSAGES     fig-FORTH )
15 ( ERROR MESSAGES     fig-FORTH )
16 ( UNUSED screens 16-19 )
17
18
19
20 ( Sound I:   Basics  fun-FORTH )
21 ( Sound II:  Noises  fun-FORTH )
22 ( Sound III: 16 bits fun-FORTH )
23 ( Sound IV:  EffectA fun-FORTH )
24 ( Sound V:   EffectB fun-FORTH )
25 ( Sound VI:  Warp    fun-FORTH )
26 ( UNUSED screens 26-29 )
27
28
29
30 ( Grfx I:   Basics   fun-FORTH )
31 ( Grfx II:  Color    fun-FORTH )
32 ( Grfx III: Turtle1  fun-FORTH )
33 ( Grfx IV:  Turtle2  fun-FORTH )
34 ( Grfx V:   Turtle3  fun-FORTH )
35 ( Grfx VI:  Boxes    fun-FORTH )
36 ( Grfx VII: Gr.1&2   fun-FORTH )
37 ( UNUSED screens 37-39 )
38
39
40 ( Special Words I    fun-FORTH )
41 ( Special Words II   fun-FORTH )
42 ( UNUSED screens 42-49 )
43
44
45
46
47
48
49
50 ( fun-FORTH intro screen        )
51
52
53
54
55
56
57
58
59
60
61
62
63
64 ( QuadraSketch  QSK    jgJUN82 )
65 ( QSK page 2           jgJUN82 )
66 ( QSK page 3           jgJUN82 )
67 ( QSK page 4           jgJUN82 )
68 ( QSK page 5           jgJUN82 )
69 ( QSK page 6           jgJUN82 )
70 ( QSK page 7           jgJUN82 )
```

8. Type

```
        1 LOAD
```

and press RETURN to load in the first screen of fun-FORTH. The screen contains, among other things, the word LOADS (see the word description later in this manual), which lets you load multiple screens.

9. Start your use of fun-FORTH by loading all its screens. Type the following commands and press RETURN after each one:

```
        2 6 LOADS (joystick, paddle, keyboard, console, and time words)
        20 25 LOADS (sound words)
        30 36 LOADS (graphics and turtle words)
        40 41 LOADS (special words)
```

<u>Note</u>. You may load parts of fun-FORTH, if you wish. Before loading a screen (using n LOAD where "n" is a screen number), list the screen (using n LIST). If the second line of the listing contains the message

```
        *** NEEDS SCREEN(S)
```

you must load in the required screen(s) first.

CREATING YOUR OWN VERSIONS OF FORTH

Follow the loading instructions above to load in the FORTH kernel and the desired packages from EXTENDED fig-FORTH and fun-FORTH. To save this custom collection on diskette for future use, insert a DOS-II formatted diskette into disk drive 1, type:

```
        SAVE
```

and press RETURN. This new copy of FORTH, like the original diskette, will load automatically. You might want to label this version "MYFORTH".

You may want to make different versions of the MYFORTH diskette later on (by loading different screens from EXTENDED fig-FORTH and fun-FORTH) for different purposes. You can copy the MYFORTH diskette by loading it into memory and then saving it on a new dikette (using the SAVE command).

SETTING UP A DISKETTE TO PROGRAM

You might want to set up a diskette for programming purposes, especially if you plan to use the EDITOR. Follow the loading instructions above and, with the fun-FORTH diskette inserted in disk drive 1, type:

```
        14 LIST MARK 15 LIST MARK
```

and press RETURN. Then insert a different DOS-II formatted diskette in disk drive 1 and type:

```
FLUSH
```

and press RETURN. These steps save two screens of error messages to the new diskette. You might want to label this diskette "MYSCREENS".

Note. You can copy the MYSCREENS diskette by using the DISKCOPY utility in fig-FORTH.

USING THE VARIOUS FORTH DISKETTES

During a normal programming session, you would load the MYFORTH diskette into memory, then insert the MYSCREENS diskette, and do your programming using the EDITOR. In this way, all screens are free, except for 14 and 15, where the error messages are.

fun-FORTH WORDS

INTRODUCTION

This section describes all the fun-FORTH screens and words. It also includes some examples of correct usage. The explanations are in the order they appear on fun-FORTH screens.

The explanations consist of the screen listing, a general explanation about the screen's contents, and the word descriptions for that screen.

Word Descriptions

The word descriptions contain the following sections:

1. The name of the word and its standard FORTH stack diagram. For example, in the diagram

```
n1  n2  --  n3
```

"--" represents the word, "n1" and "n2" are parameters on the stack prior to execution of the word, and "n3" is a parameter returned by the word after its execution. The rightmost parameters are on top of the stack. The parameters in the stack diagram usually have meaningful names, such as "joystick#", instead of "n1". If a word has no parameters, the stack diagram is " -- " .

2. An example of a simple, standard use of the word, with values that correspond to the parameters in the stack diagram. For words that return values, like STICK (a joystick word), a period (.) followed by a number and the prompt "ok" are used to represent the printing of the returned value. If a word has no parameters, the example line contains the phrase "no parameters".

3. A description of the word.

4. Warnings and comments that may help debugging and program design.

5. An additional example of word usage. Use these examples as a starting point for your own experimentation. Remember to load the screen on which the word appears.

SCREEN #1

```
SCR # 1
  0 ( fun-FORTH by Joel Gluck 1982 )
  1 ( Type 50 LOAD for intro. )
  2 : LOADS 1+ SWAP DO I LOAD LOOP ;
  3 ( firstscreen lastscreen --   )
  4 : ARRAY <BUILDS 1+ DUP + ALLOT
  5   DOES> SWAP DUP + + ;
  6 ( to create: size -- XXX      )
  7 ( to store: val index  XXX ! )
  8 ( to retrieve:  index XXX @   )
  9 ( to print:  index XXX ?      )
 10 ( range:  0 <= index <= size )
 11 : RND -11766 C@ -11766 C@ 256 *
 12   + SWAP MOD ABS ;
 13 ( range -- random# )
 14 : DELAY < DO 10 0 DO LOOP LOOP  ;
 15 ( length -- ) ;S
```

Screen #1 contains some of the most frequently used words in fun-FORTH.

LOADS firstscreen lastscreen --

Example: 20 25 LOADS

Loads multiple FORTH screens from diskette, starting with firstscreen and ending with lastscreen.

Warning. Don't use on EXTENDED fig-FORTH screens connected to one another by "-->". Just load the first one in those cases.

ARRAY

Example: 5 ARRAY MYGRADES

Creates an array named MYGRADES capable of storing 6 values (0-5).

To create an array: size ARRAY XXX

To store a value: value index XXX !

To retrieve a value: index XXX @

To print a value: index XXX ? (where index ranges from 0 to size.)

RND   range -- random#

Example: 5 RND . 2 ok

Returns a random number from 0 to range minus 1.

Comment. If a random byte is needed (0-255), use -11766 C@ . It's faster.

DELAY   length --

Example: 900 DELAY

Produces a pause in execution for a time proportional to length.

Comment. 10000 DELAY equals about 13 seconds.

SCREEN #2

```
SCR # 2
( Joystick Words    fun-FORTH   )
: STICK 632 + C@ ;
( joystick# -- value  <*BASIC* )
: STIG 644 + C@ ;
( joystick# -- 0/1    <*BASIC* )

  0 VARIABLE JOYXY -2 ALLOT 1 ,
  -1 , 0 , 0 , 1 , -1 , 0 , 0 ,
  1 , -1 , 0 , 1 , 1 , 1 , 0 ,
  -1 , -1 , -1 , 0 , 0 , 0 , 0 ,
: XSTK STICK 6 + DUP +
  JOYXY + @ ;
( joystick# -- -1/0/1 )
: YSTK STICK 5 - DUP +
  JOYXY + @ ;
( joystick# -- -1/0/1 ) ;S
```

Screen #2 contains all four joystick words.

STICK   joystick# -- value

Example: 0 STICK . 15 ok

Similar to the STICK() function in ATARI BASIC, this word returns the value of the specified joystick. Refer to page 60 of the <u>BASIC Reference Manual</u> for the directional meaning of the value.

STRIG   joystick# -- 0/1

Example: 3 STRIG . 1 ok

Similar to the STRIG() function in ATARI BASIC, this word returns the trigger value of the specified joystick. Trigger down returns 0, and trigger up returns 1.

<u>Additional example</u>. The following word waits until the trigger of the leftmost joystick is pressed:

```
: TRIGWAIT BEGIN 0 STRIG 0= UNTIL ; ( -- )
TRIGWAIT
```

XSTK   joystick# -- -1/0/1

Example: 1 XSTK . -1 ok

Returns a value representing the horizontal component of the direction in which the specified joystick is being pulled. Left=-1, middle=0, and right=1.

Comments. In combination with YSTK, this word is every game programmer's (and most other programmers') dream come true. In the past (especially in ATARI BASIC), a routine had to be written that either looked up the directional information in an array or figured it out with a series of IF/THEN statements. But with these fun-FORTH words, writing a simple joystick drawing program is really simple.

Also notice that the directional values returned do match the direction of X-Y coordinate growth on the screen.

Another advantage of using XSTK instead of STICK in a program needing only horizontal values is that XSTK also translates any accidental diagonals into the correct direction.

YSTK   joystick# -- -1/0/1

Example: 2 YSTK . 0 ok

The vertical counterpart to XSTK, this word returns the vertical component of the direction in which the specified joystick is being pulled. Up=-1, middle=0, and down=1.

Comment. See the comments under XSTK.

SCREEN #3

```
SCR # 3
  0 ( Paddle Words           fun-FORTH )
  1
  2 : PADDLE 624 + C@ ;
  3 ( paddle# -- value        <*BASIC* )
  4
  5 : PTRIG 636 + C@ ;
  6 ( paddle# -- 0/1          <*BASIC* )
  7
  8   ;S
  9
 10
 11
 12
 13
 14
 15
```

Screen #3 contains the two paddle words.

PADDLE   paddle# -- value

Example: 0 PADDLE . 210 ok

Similar to the PADDLE( ) function in ATARI BASIC, this word returns the value of the specified paddle. This value ranges from 1 to 228 . For more information about this function, see page 59 of the BASIC Reference Manual.

PTRIG   paddle# -- 0/1

Example: 7 PTRIG . 0 ok

Similar to the PTRIG() function in ATARI BASIC, this word returns the trigger value of the specified paddle. Trigger down returns 0, and trigger up returns 1.

SCREEN #4

```
SCR # 4
( Keyboard Words     fun-FORTH )

: KEYINIT 255 764 C! ;
( --     <clears 1-key buffer )

: KEYBD? 764 C@ 255 < ;
( -- 0/1 )

: PAKTC KEYINIT BEGIN KEYBD?
  UNTIL KEYINIT ;
( --     <Push Any Key To Cont. )

: Y/N KEYINIT 0 BEGIN DROP
  KEY DUP 89 = DUP ROT 78 =
  OR UNTIL ;
( -- 0/1 ) ;S
```

Screen #4 contains four useful words to aid in keyboard I/O.

KEYINIT --

no parameters

Used in conjunction with KEYBD? , this word clears the one-key buffer.

KEYBD? -- 0/1

Example: KEYBD? . 1 ok

Returns a 1 (true) if a key was pressed or a 0 (false) if a key wasn't pressed.

<u>Comments</u>. It's sometimes useful to execute KEYINIT before and after using KEYBD? to clear the buffer.

Note that KEYBD? operates "on the fly". That is, it won't delay further execution.

PAKTC --

no parameters

This word waits until any key (excluding SHIFT, CNTRL, BREAK, and the console keys) is pressed and then allows execution to continue.

<u>Comments</u>. The difference between PAKTC and KEY DROP is that PAKTC is silent (that is, no key click is produced).

PAKTC stands for Push Any Key To Continue.

Y/N -- 0/1

Example: Y/N . 0 ok

This word waits until a "Y" or an "N" is pressed. A "Y" returns a 1 (true), and an "N" returns a 0 (false). Any other key pressed has no effect.

<u>Comment</u>. Use of this word usually follows a "yes/no" question posed on the screen. Note that this word doesn't echo the typed character on the screen.

SCREEN #5

```
SCR # 5
( Console Words        fun-FORTH )

: CONSOLE? 53279 C@ ;
( -- value )

: START? CONSOLE? 2 MOD 0= ;
( -- 0/1 )

: SELECT? CONSOLE? 2 / 2 MOD 0= ;
( -- 0/1 )

: OPTION? CONSOLE? 4 / 2 MOD 0= ;
( -- 0/1 ) ;S



```

Screen #5 contains the four words affecting the yellow console keys on the far right-hand side of the ATARI Computer keyboard.

CONSOLE?   -- value

Example: CONSOLE? . 7 ok

Returns the value of the CONSOL byte (location 53279).

START?   -- 0/1

Example: START? . 1 ok

Returns a 1 if the START key is being pressed, and a 0 if it isn't.

Additional example. The following word waits until the START key is pressed:

```
: STWAIT BEGIN START? UNTIL ; ( -- )
STWAIT
```

SELECT?   -- 0/1

Example: SELECT? . 0 ok

Returns a 1 if the SELECT key is being pressed, and a 0 if it isn't.

OPTION?   -- 0/1

Example: OPTION? . 1 ok

Returns a 1 if the OPTION key is being pressed, and a 0 if it isn't.

SCREEN #6

```
SCR # 6
( Time Words          fun-FORTH )

: SECONDS 20 C@ S->D 19 C@ 256
  M* D+ 60 M/ SWAP DROP ;
( -- #secs )

: SETIME 60 * 256 /MOD 19 C!
  20 C! ;
( #secs -- ) ;S
```

Screen #6 contains two useful words to perform timing (in seconds).

SECONDS   -- #seconds

Example: SECONDS . 59 ok

Returns the current time (originally set by SETIME) in seconds.

SETIME   #seconds --

Example: 0 SETIME

Sets the current time (in seconds).

Comment. The most common use of SETIME is 0 SETIME, although other values are useful to continue timing after some break in the action.

Additional example. The following word pauses for the supplied number of seconds:

```
: TPAWZ 0 SETIME BEGIN DUP SECONDS = UNTIL DROP ; ( sec -- )
10 TPAWZ
```

SCREEN #20

```
SCR # 20
( Sound I:   Basics  fun-FORTH )
: FILTER! 53768 C! ; ( n -- )
: SINIT 3 53775 C! 3 562 C!
  0 FILTER! ;
( --     <initializes for sound )
: SOFF DUP + 0 SWAP 53761 + C! ;
(voc -- )
: ASOFF 4 0 DO I SOFF LOOP ;
( --     <shuts off all voices )

: SND SWAP 16 * + ROT DUP +
  53760 + ROT OVER C! 1+ C! ;
( voc frq tim vol -- )
( called SOUND in *BASIC* )
: FREQ SWAP DUP + 53760 + C! ;
( voc frq -- ) ;S
```

Screen #20 contains the basic sound words. These take the place of the original sound words supplied with EXTENDED fig-FORTH, which you shouldn't load.

FILTER!   n --

Example: 0 FILTER!

Pokes a value into the AUDCTL (audio-control) byte. For more information about AUDCTL, see page 7-9 of De Re ATARI (APX-90008).

Comment. FILTER! is originally from EXTENDED fig-FORTH and is transcribed here for convenience.

SINIT   --

no parameters

Initializes for sound use. Re-execute after any external I/O, including diskette access.

SOFF   voc --

Example: 1 SOFF

Shuts off the specified sound voice.

Comment. SOFF is faster than SND in turning off sound. SOFF shuts off only the AUDC byte, leaving AUDF intact. For more information about AUDC and AUDF, see pages 7-2 and 7-3 of De Re ATARI.

ASOFF --

no parameters

Shuts off all of the sound voices.

<u>Comment</u>. ASOFF can be a great relief!

SND voc frq tim vol --

Example: 0 100 10 8 SND

Similar to the SOUND command in ATARI BASIC, this word creates and sustains a sound based on the specified voice (0-3), frequency (pitch) (0-255), timbre (distortion) (0-14 even), and volume (0-15). For more information about SND, see page 57 of the <u>BASIC Reference Manual</u>.

<u>Comment</u>. SND takes the place of the sound command in EXTENDED fig-FORTH.

FREQ voc frq --

Example: 1 255 FREQ

Changes the frequency of the specified voice to a new specified frequency. It doesn't effect the timbre or volume.

<u>Comment</u>. FREQ is significantly faster than SND for changing just the frequency.

<u>Additional example</u>. The following word makes an "alien attack" type sound effect.

```
: ALIEN 0 0 10 15 SND 10 0 DO 256 0 DO 0 I FREQ LOOP LOOP 0 SOFF ;
ALIEN
```

SCREEN #21

```
SCR # 21
( Sound II:  Noises  fun-FORTH )
( *** NEEDS SCREENS: 1,20 )
: BOOP 0 SWAP 10 8 SND 50 DELAY
  0 SOFF ; ( frq -- )
: BUZZ 0 0 4 15 SND 125 DELAY
  0 SOFF ; ( -- )
: BELL 0 15 DO DUP 0 SWAP 10 I
  SND 30 DELAY -1 +LOOP DROP
  0 SOFF ; ( frq -- )
: EXPLODE SWAP 0 15 DO 2DUP
  0 SWAP 8 I SND DELAY -1
  +LOOP 2DROP 0 SOFF ;
( frq dur -- )
: CLICK 0 53279 C! ; ( -- )
: SPOP 31 53761 C! 5 0 DO LOOP
  0 53761 C! ; ( -- ) ;S
```

Screen #21 contains six useful and interesting noise effects that can make any program (game or serious) more enjoyable to use.

BOOP frq --

Example: 150 BOOP

Produces a short, pure tone at the desired frequency.

Warning. BOOP essentially performs a 0 SOFF. It interferes with use of voice zero.

Comment. You can use BOOP for anything--a ball bouncing against a wall, a unique keyboard feedback, you name it.

BUZZ --

no parameters

Produces a sharp, loud, beep noise (BEEP is a different, EXTENDED fig-FORTH word using the console speaker).

Warning. See the warning under BOOP.

Comment. BUZZ is great for error warnings and the like.

BELL frq --

Example: 100 BELL

Produces a bell-like sound of the desired frequency.

Warning. See the warning under BOOP.

Comment. BELL is useful for acknowledging good input.

EXPLODE   frq dur --

Example: 50 120 EXPLODE

Produces an explosion noise at the desired frequency for a length of time proportional to dur.

Warning. See the warning under BOOP.

Comment. You could write a more useful word to produce the sound and a desired graphic effect simultaneously.

CLICK   --

no parameters

Produces a small click from the console speaker.

Comment. Many clicks in rapid succession produce an interesting buzz.

Additional example. The following word produces a buzz from the console speaker for the desired length.

```
: BUZZ2 0 DO CLICK LOOP ; ( n -- )
2000 BUZZ2
```

SPOP   --

no parameters

Produces a single "pop" from the TV speaker.

Warning. SPOP interrupts use of voice zero.

Comment. Many pops in succession with different small pauses in between can produce a wide range of unique noises. For more informationm see page 7-8 of De Re ATARI.

Additional example. The following word produces a noise based on the supplied pause length.

```
: NOIZ 300 0 DO DUP SPOP 0 DO LOOP LOOP ; ( dur -- )
10 NOIZ
```

SCREEN #22

```
SCR # 22
( Sound III: 16 bits fun-FORTH )
( *** NEEDS SCREEN: 20 )
: SINIT16 SINIT 120 FILTER!
  ASOFF ;
( --     <init. for 16-bit snd )

: SND16 SWAP 16 * + ROT DUP 4 *
  53763 + ROT SWAP C! SWAP 256
  /MOD ROT 4 * 53762 + DUP ROT
  SWAP C! 2 - C! ;
( voc0or1 frq16 tim vol -- )

: FREQ16 256 /MOD ROT 4 *
  53762 + DUP ROT SWAP C!
  2 - C! ;
( voc0or1 frq16 -- ) ;S
```

Screen #22 supplies you with a much wider sound range. In normal sound, frequency is defined by 8 bits (1 byte). The words on this screen allow a frequency defined by 16 bits (2 bytes). The advantages are much lower and higher frequencies than previously available and much finer increments between frequencies. The only disadvantage is that you can work with only two voices instead of four. For more information, see pages 7-10 and 7-11 of De Re ATARI.

SINIT16 --

no parameters

Initializes for 16-bit sound.

Warning. SINIT16 interrupts normal sound use.

SND16 voc0or1 frq16 tim vol --

Example: 0 3000 10 8 SND16

Similar to SND except for two differences:
1) There are only 2 voices (0 and 1).
2) Frequency (frq16) ranges from 0 to 65280.

Warning. SINIT16 must be executed at least once before using SND16 or FREQ16.

Additional examples.

0 30000 10 15 SND16 (very low pure tone)
0 0 8 15 SND16 (very fine white noise)
ASOFF (to shut it off)

FREQ16   voc0or1 frq16 --

Example: 0 3200 FREQ16

Similar to FREQ but with only two voices and a 16-bit frequency range.

Warning. See the warning under SND16.

Additional example. The following word loops from 1000 to 1500 as frequencies.

```
: TEST16 0 0 10 8 SND16 1500 1000 DO 0 I FREQ16 LOOP ; ( -- )
SINIT16 16 TEST ASOFF
```

SCREEN #23

```
SCR # 23
( Sound IV:  EffectA fun-FORTH )
( *** NEEDS SCREEN: 20 )

: SFXA1 24 FILTER! 160 +
  53768 53761 DO DUP I C!
  2 +LOOP DROP ;
( vol --    <init. effect A )

: SFXA2 53760 C! 53764 C!
  53762 C! 53766 C! ;
( frq frq frq frq -- ) ;S





```

Screen #23 contains the two SFXA (sound effect A) words. Using them, you can reproduce a wide range of sound effects (about 4.3 billion different ones!) that are all rhythmic and unusual. An extra advantage over normal methods is that the effect requires no maintenance; turn it on and it keeps going (until you shut it off). For a more detailed look at the method used, see page 7-12 of De Re ATARI.

SFXA1 vol --

Example: 10 SFXA1

Initializes for sound effect A using the specified volume.

SFXA2 frq frq frq frq --

Example: 10 9 255 50 SFXA2

Produces unusual sound effects based on the four frequencies provided.

Warning. SFXA1 must be executed at least once before using SFXA2. Also, use of these words interrupts normal sound.

Comment. You can easily shut off any SFX with an ASOFF or SINIT followed by new sounds.

Additional examples. The following examples only scratch the surface of what this word can do.

```
SINIT 12 SFXA1  (execute these first)
8 8 254 255 SFXA2
4 5 0 0 SFXA2
```

```
30 1 155 0 SFXA2
1 1 254 255 SFXA2
48 48 155 154 SFXA2
54 55 100 99 SFXA2
49 50 0 0 SFXA2
1 any# 0 255 SFXA2
ASOFF (shuts it off)
```

SCREEN #24

```
SCR # 24
( Sound V:   EffectB fun-FORTH )
( *** NEEDS SCREEN: 20 )

: SFXB1 4 FILTER! ASOFF 160 +
  DUP 53761 C! 53765 C! ;
( vol --    <init. effect B )

: SFXB2 53760 C! 53764 C! ;
( frq frq -- ) ;S
```

Screen #24 contains the two SFXB words (sound effect B). These produce simpler (as compared to SFXA), yet quite unique, sounds.

SFXB1 vol --

Example: 10 SFXB1

Initializes for sound effect B using the specified volume.

SFXB2 frq frq --

Example: 255 254 SFXB2

Produces unusual sound effects based on the two frequencies provided.

Warning. SFXB1 must be executed at least once before using SFXB2. Also, use of these words interrupts normal sound.

Comment. You can easily shut off any SFX with an ASOFF or SINIT followed by new sounds.

Additional examples. The following examples are a few of the more interesting SFXB2 combinations.

```
SINIT 12 SFXB1 (execute these first)
254 255 SFXB2
150 151 SFXB2
24 101 SFXB2
100 200 SFXB2
51 101 SFXB2
```

82 98 SFXB2
ASOFF (shuts it off)

SCREEN #25

```
SCR # 25
( Sound VI:  Warp    fun-FORTH )
( *** NEEDS SCREENS: 1,20 )

: WARP SWAP 16 * + 53761 C! SWAP
  2DUP - 0< IF -1 ELSE 1 ENDIF
  ROT ROT DO 0 I FREQ SWAP DUP
  DELAY SWAP DUP +LOOP 2DROP ;
( dur frq1 frq2 tim vol -- )

  ;S
```

Screen #25 contains the word WARP. WARP is neither very fast nor very useful, but it does show a simple, general purpose example of sound use.

WARP   dur frq1 frq2 tim vol --

Example: 5 0 255 10 15 WARP

Produces a sound using the specified timbre and volume, which loops from frq1 to frq2 at a rate inversely proportional to dur (that is, 0 is fastest for dur). The last frequency remains on after execution and may be shut off with a 0 SOFF .

Warning. WARP interrupts use of voice zero. Also, see the warning under SINIT.

Comments. The operand frq1 may be higher or lower than frq2, which gives the word greater versatility.

The last sound remains on to allow continuity with another WARP or other sounds.

A faster version of WARP dedicated to producing one warp-like sound can easily be written.

Additional example. The following word repeats a certain WARP the specified number of times.

```
: RED-ALERT 0 DO 10 130 100 10 15 WARP LOOP 0 SOFF ; RTN
SINIT 10 RED-ALERT
```

SCREEN #30

```
SCR # 30
( Grfx I:   Basics   fun-FORTH )
( *** NEEDS: graphics screens
( from fig-FORTH:50-55;see doc.)

: FGR &GR 12 MASK ! GR. ;
( mode --      <full-screen GR. )
: SGR &GR 28 MASK ! GR. ;
( mode --  <use instead of GR. )
  HEX CODE CGET XSAVE STX, # 30
  LDX, # 7 LDA, IOCX 2 + ,X STA,
  TYA, IOCX 8 + ,X STA, IOCX 9 +
  ,X STA, CIO JSR, XSAVE LDX,
  PUSH0A JMP, DECIMAL ( -- )
: SCR? POS CGET ;
( x y -- color#/ascii# )
( called LOCATE in *BASIC* ) ;S
```

Screen #30 contains some of the basic graphics words. These words let you get a full screen of graphics in FORTH (using FGR). Also, you no longer have to exit a graphics mode (using EXTENDED fig-FORTH's XGR word) before going into a different mode; FGR and SGR take care of the details for you. In addition, with SCR? you can do the equivalent of an ATARI BASIC LOCATE command, which "looks" at the screen and returns color numbers or ASCII numbers.

FGR   mode --

Example: 5 FGR

Similar to the GR. command in ATARI BASIC and EXTENDED fig-FORTH, this word sets up the full-screen version of the desired mode.

Warning. Don't add 16 to the mode number as you would in ATARI BASIC for a full screen.

Comment. A 0 FGR lets you plot and draw on the mode 0 screen. An XGR does not.

SGR   mode --

Example: 2 SGR

Similar to FGR, this word sets up the split-screen version of the desired mode, with four lines of graphics mode zero on the bottom.

Comment. This word is preferred to GR. because it can always get you out of the current mode and into a new one.

SCR?   x y -- color#/ascii#

Example: 22 15 SCR? .   2 ok

Similar to the LOCATE command in ATARI BASIC, this word returns the color# (in modes 3 and up) or the ascii# (in modes 0, 1, & 2) of the specified coordinates on the screen.

<u>Warning</u>. Using this word in mode zero may affect the display and move the cursor.

<u>Comment</u>. Here's an analogy for those who don't know what LOCATE does:

SCR? is to PLOT as @ is to !

SCREEN #31

```
SCR # 31
( Grfx II:  Color    fun-FORTH   )
( *** NEEDS SCREEN: 30
  1 VARIABLE COL
: TCOL COL ! ; ( color -- )
(called COLOR in *BASIC* )

: TCOL? COL @ ; ( -- color )

: CPLOT TCOL? ROT ROT PLOT ;
(x y -- )
( called PLOT in *BASIC* )

: CDRAW TCOL? ROT ROT DRAW ;
( x y -- )
(called DRAWTO in *BASIC* ) ;S

```

Screen #31 contains the basis for the color words: the variable COL (plus TCOL and TCOL? , which access it). Using COL , words have been written that use the stored color. Among these are CPLOT and CDRAW (on this screen), CBOX and CHBOX (on screen #35), and all of the turtle words that do any drawing (see screens #32-#34).

TCOL   color# --

Example: 2 TCOL

Store the specified color for later use. Similar to COLOR in ATARI BASIC.

Comment. If you're in a text mode, you should specify an ascii# instead of a color#.

TCOL?   -- color#

Example: TCOL? . 2 ok

Returns the current color being stored for use with the turtle and color words (CPLOT, etc.). See TCOL.

CPLOT   x y --

Example: 22 15 CPLOT

Similar to PLOT in ATARI BASIC, this word plots a point at the specified coordinates using the stored color (see TCOL).

Comment. In most cases, this word, in combination with TCOL, is more convenient than PLOT (which requires the color parameter each time).

CDRAW   x y --

Example: 32 25 CDRAW

Draws a line from the last two coordinates to the specified coordinates using the stored color.

Comment. See the comment under CPLOT.

SCREEN #32

```
SCR # 32
( Grfx III: Turtle1  fun-FORTH )
( *** NEEDS SCREEN: 31 )
 10 VARIABLE TX 10 VARIABLE TY
  0 VARIABLE XD -1 VARIABLE YD
  1 VARIABLE DIR
  0 VARIABLE TURTD -2 ALLOT
  0 , -1 , 1 , -1 , 1 , 0 , 1 ,
  1 , 0 , 1 , -1 , 1 , -1 , 0 ,
  -1 , -1 ,
: TXY? TX @ TY @ ; ( -- tx ty )
: TPLOT TXY? CPLOT ; ( -- )
: TF TPLOT YD @ TY +! XD @
  TX +! ; ( -- )
: TFWD 0 DO TF LOOP ; ( n -- )
: TMOVE TY ! TX ! ; ( x y -- )
  ;S
```

Screen #32 is the first "turtle graphics" screen. The simplified turtle presented on this and the following two screens is reminiscent of the turtle in WSFN (APX-20026). It, too, can't handle any angles, but instead only 45-degree increments (vertical, horizontal, and the four diagonals). The fun-FORTH turtle is even more simple in that it never appears on the screen; you have to keep track of its direction and coordinates.

The turtle has a direction (1-8, see TDIR on the next screen), x and y coordinates, and a color to draw with (stored by TCOL). Using the commands on these three screens, you can make the turtle do whatever your heart desires: draw lines, turn around, draw simple shapes, jump around the screen, or even play "Breakout" (although that one takes a little work).

TXY?   -- tx ty

Example: TXY? . . 20 40 ok

Returns the current coordinates of the turtle.

TPLOT   --

no parameters

Plots a point directly underneath the turtle using the stored color.

Comment. TPLOT is a quick way to determine where the turtle is during experimentation.

TF   --

no parameters

Moves the turtle one pixel in the stored direction (see TDIR) and plots a point where it used to be using the stored color.

Comment. No point is plotted at the turtle's new position.

TFWD   n --

Example: 5 TFWD

Moves the turtle n pixels in the stored direction, each time plotting a point where it has been, using the stored color.

Comment. No point is plotted at the turtle's final position.

TMOVE x y --

Example: 45 25 TMOVE

Jumps the turtle to the specified coordinates.

Comment. This word doesn't plot any points or change the display in any way.

SCREEN #33

```
SCR # 33
( Grfx IV:  Turtle2  fun-FORTH)
( *** NEEDS SCREEN: 32 )

: TDIR DUP DIR ! DUP DUP + 2
  - DUP + TURTD + @ XD ! DUP
  + 1 - DUP + TURTD + @ YD ! ;
( direction -- )
: TDIR? DIR @ ;
( -- direction )
: TTN TDIR? 1+ DUP TDIR 9 =
  IF 1 TDIR ENDIF ;
( --        <one turn clockwise )
: TURN 0 DO TTN LOOP ;
( n --      <n turns clockwise )
: TGO DUP YD @ * TY +! XD @
  * TX +! ; (n -- ) ;S
```

Screen #33 is the second screen of turtle graphics words.

TDIR direction# --

Example: 5 TDIR

Stores the specified direction (from 1 to 8) for use in turtle movement (TF, TFWD, and TGO). Directions go clockwise from 1 (north) to 8 (northwest).

Comment. Here's a diagram of turtle directions:

```
   UP
8  1  2
 \ | /
7--*--3
 / | \
6  5  4
```

TDIR? -- tdirection#

Example: TDIR? . 4 ok

Returns the current turtle direction number.

TTN --

no parameters

Rotates the turtle one direction# clockwise (for example, from 4 to 5).

TURN n --

Example: 3 TURN

Rotates the turtle n direction numbers clockwise.

Comment. A 2 TURN is a 90-degree (clockwise) turn.

TGO n --

Example: 5 TGO

Moves the turtle n pixels in the stored direction but doesn't plot any points.

```
SCR # 34
( Grfx V   Turtle3   fun-FORTH )
( *** NEEDS SCREEN: 33 )
: TSCR? TX @ TY @ SCR? ;
( -- color )
: TAHD? TX @ XD @ + TY @ YD @
  + SCR? ; ( -- color )
: TDRAW TPLOT 2DUP CDRAW
  TMOVE ; ( x y -- )
: GR? 87 C@ ; ( -- mode )
: HOME 1 TCOL 1 TDIR GR? >R
  R 3 = IF 20 10 TMOVE ENDIF
  R 3 > IF 40 20 TMOVE ENDIF
  R 5 > IF 80 40 TMOVE ENDIF
  R 7 > IF 160 80 TMOVE ENDIF
  R> DROP ;
( --     <homes turtle ) ;S
```

Screen #34 is the last of the three turtle screens. Unlike the other two screens, the words on this screen are turtle conveniences, not turtle essentials.

TSCR? -- color#

Example: TSCR? . 1 ok

Returns the color# of the point directly beneath the turtle.

TAHD? -- color#

Example: TAHD? . 3 ok

Returns the color# of the pixel directly ahead of the turtle, that is, one pixel forward in the stored direction.

TDRAW x y --

Example: 10 15 TDRAW

Draws a line using the stored color from the turtle's position to the specified coordinates, and moves the turtle to those coordinates.

GR? -- mode

Example: GR? . 7 ok

Return the current mode number.

Warning. This word may have no practical use at all, except that (1) it's great for amnesiac programmers and (2) it was used in defining the word HOME.

HOME --

no parameters

Moves the turtle to the appropriate center of the screen, sets its direction to 1 (up) and its color to 1.

Warning. This word isn't for modes 0, 1, 2, or greater than 8 (available with the GTIA chip).

Comment. It's convenient to execute HOME upon entering a mode before playing with the turtle.

USING THE TURTLE WORDS

One nice way to play with the turtle words is to go into mode 5 or 7 with SGR, do a HOME, and do a few direct turtle commands to get the hang of it. Then, start to define simple words (while you're still in the text window) that make the turtle draw simple shapes and patterns. Finally, you'll find that you're defining larger words that draw beautiful multicolored pictures or patterns on the screen. Not only is this approach fun, but it's a great way to introduce children to FORTH.

SCREEN #35

```
SCR # 35
( Grfx VI:  Boxes    fun-FORTH )
( *** NEEDS SCREEN: 31 )

: CBOX 1+ SWAP DO 2DUP I CPLOT
  I CDRAW LOOP 2DROP ;
( xlow xhi ylow yhi -- )

: CHBOX >R >R DUP CPLOT SWAP
  DUP R CDRAW R> ROT ROT R
  CDRAW DUP R> CDRAW SWAP
  CDRAW
( xlow xhi ylow yhi -- )
( CHBOX is a hollow box ) ;S
```

Screen #35 contains two very useful words: one draws a filled box, and the other draws a hollow box. These words can be as much fun to play with as the turtle words. (Note. When you experiment with these words, use graphics mode 3 or greater by executing, for example, 3 SGR .)

CBOX xlow xhi ylow yhi --

Example: 12 26 5 10 CBOX

Draws a solid box using the stored color (see TCOL) whose corners have the following coordinates: (xlow,ylow) (xhi,ylow)(xhi,yhi) and (xlow,yhi).

Warning. xhi must be greater than xlow and yhi must be greater than ylow.

CHBOX xlow xhi ylow yhi --

Example: 12 26 5 10 CHBOX

Draws a hollow box using the stored color whose corners have the specified coordinates (as with CBOX).

```
SCR # 36
( Grfx VII: Gr.1&2   fun-FORTH )
( *** NEEDS SCREEN: 30 )
: GSPACE BL CPUT ;
: GSPACES 0 MAX -DUP IF 0 DO
  GSPACE LOOP ENDIF ; ( n -- )
: GD.R >R SWAP OVER DABS <#
  #S SIGN #> R> OVER -
  GSPACES GTYPE ; ( d fldwth - )
: G.R >R S->D R> GD.R ;
( n fieldwidth -- )
: GD. 0 GD.R ; ( d -- )
: G. S->D GD. ; ( n -- )
( . for modes 1 & 2. Note: G.
( doesn't follow #s with blanks)
: GCR 155 CPUT ; ( -- )
( CR for modes 1 & 2 ) ;S
```

Screen #36 conains words that aid in printing on the graphics mode 1 and 2 screens. The most needed of the words are G. (which prints the top of the stack onto the screen) and GCR (which does a carriage return to the next line).

GSPACE --

no parameters

Outputs a space to the mode 1 or 2 screen.

GSPACES n --

Example: 4 GSPACES

Outputs n spaces to the mode 1 or 2 screen.

GD.R d fieldwidth --

Example: 294.863 7 GD.R

Outputs the double-precision number, right-justified in the specified field width, to the mode 1 or 2 screen.

G.R n fieldwidth --

Example: 1215 5 G.R

Outputs the number, right-justified in the specified field width, to the mode 1 or 2 screen.

Comment. G.R is a useful word for printing scores or times on the screen.

GD. d --

Example: 294.863 GD.

Outputs the double-precision number to the mode 1 or 2 screen.

G. n --

Example: 1215 G.

Outputs the number to the mode 1 or 2 screen.

Comment. This is essentially a period (.) for the large text modes, the major difference being that it doesn't follow the number with a space.

GCR --

no parameters

Outputs a carriage return to the mode 1 or 2 screen.

Comment. GCR is a carriage return (CR) for the large text modes.

SCREEN #40

```
SCR # 40
( Special Words I  fun-FORTH )
: MYSELF LATEST PFA CFA
  , ; IMMEDIATE
( --     <recursive word )
: PICK 1+ DUP + SP@ + @ ;
( stackdepth -- contents )
: CVARIABLE <BUILDS 1 ALLOT
  DOES> ;
( to create:  CVARIABLE XXX )
( note: no initial value nec. )
( use C! C@ and C? with XXX )
: CARRAY <BUILDS 1+ ALLOT
  DOES> + ; ( size CARRAY XXX )
( to store: val index XXX C! )
( to retrieve: index XXX C@ )
( range : 0 <= index <= size ) ;S
```

Screen #40 is the first of two special words screens, containing miscellaneous useful words. The words on Screen #40 screen have appeared in various FORTH sources (such as FORTH Dimensions, the FORTH Interest Group publication. See the EXTENDED fig FORTH bibliography).

MYSELF --

no parameters

Permits recursion in FORTH. Use MYSELF in a definition when you want to call the word you're defining.

Warning. If you're not familiar with recursion, don't use MYSELF; it easily bombs the system.

Additional example. The following word prints items from the stack until it encounters a zero:

```
: MYTEST DUP 0= IF ." done!" ELSE CR . MYSELF ENDIF ;
( 0 n n n n n -- 0 )
0 233 -5 1290 771 MYTEST
```

PICK stackdepth -- contents

Example: 43 19 29 2 PICK . 43 ok

Returns the number from the desired depth of the stack.

Warning. Stack depth should not exceed the actual depth of the stack; if it does, the result is garbage.

Comment. A 0 PICK is the same thing as a DUP. A 1 PICK is the same thing as an OVER.

CVARIABLE   -- XXX

Example: CVARIABLE MYVAR

Defines a one-byte variable with the specified name, and no initial value. Use C! , C@ , and C with the created variable.

CARRAY

Example: 5 CARRAY MYBYTES

Creates a one-byte-per-element array named MYBYTES capable of storing six values (0-5). Use C! , C@ , and C? with the created array. See ARRAY (screen #1) for more information.

SCREEN #41

```
SCR # 41
( Special Words II   fun-FORTH )

: ATTRON 128 77 C! ; ( -- )
: ATTROFF 0 77 C! ; ( -- )
( turn attract mode on & off )

: CURSON 0 752 C! ; ( -- )
: CURSOFF 1 752 C! ; ( -- )
( turn cursor on & off )

: ANTON 34 559 C! ; ( -- )
: ANTOFF 0 559 C! ; ( -- )
( turn antic on & off ) ;S



```

Screen #41 contains six words that turn things on and off with a single poke. These are nice for programmers who can never seem to remember the right addresses or values.

ATTRON --

no parameters

Turns the "attract mode" on.

ATTROFF --

no parameters

Turns the "attract mode" off and ensures that it will stay off for about nine minutes.

Comment. ATTROFF is useful for applications in which the keyboard is not in use.

CURSON --

no parameters

Turns the mode 0 cursor (white box) on.

CURSOFF --

no parameters

Turns the mode 0 cursor (white box) off.

Comment. CURSOFF makes output in mode 0 look more graceful (no racing cursor).

ANTON --

no parameters

Turns the ANTIC chip (the display chip) "on".

ANTOFF --

no parameters

Turns the ANTIC chip "off". The screen goes black, but all else is normal.

Comment. Turning ANTIC off can increase operational speed approximately 30 percent over the operational speed while ANTIC is on and you're in mode 0, because ANTIC isn't stealing cycles from the 6502 microprocessor.

TROUBLESHOOTING

DRIFTING OF TIME VALUE

When using SECONDS (which has a ceiling of about 18 minutes), after about the 550th second the value will be off by four seconds. If you find a fix to this, please send it along.

READJUSTING SOUND PROBLEMS

When using sound words, if anything doesn't sound quite right (including the SFX's), try executing SINIT again (or SINIT16, for 16-bit sound), and re-execute your sound code. This usually works.

MOVING THE TURTLE OFF THE SCREEN

When using turtle graphics, moving offscreen doesn't cause errors. However, if you go far enough off the screen, the turtle wraps around back onto the screen again.

COMPUTER LOCKUP

If the computer locks up while you're testing a new word, the chances are good the word was overflowing the stack. Try pushing the SYSTEM RESET button. If this doesn't work, shut off the system and reload the FORTH diskette.

ADVANCED TECHNICAL INFORMATION

MASKING OUT WORDS

Sometimes you'll want to use only some of the words from a particular fun-FORTH screen. In these instances, you can mask out all the unwanted words on a screen. However, before doing so, make sure any other words you plan to load don't depend on any words you intend to mask out. For example, suppose you don't want the SND command on Screen #20 but you do want BUZZ and BELL on Screen #21. Checking the definitions of BUZZ and BELL, you'll see they both need the SND command. Therefore, you must load SND if you want BUZZ and BELL.

Some words are even trickier in that they don't contain the words they depend on. An example of this is the SINIT word; almost all sound words need SINIT before they can be used, but these words do not necessarily contain SINIT in their definitions.

To mask out unwanted words, go into the EDITOR and enclose in parentheses the lines in which the words are contained. Every line containing the definition of an unwanted word should be enclosed. (If you don't know how to use the EDITOR, see the EXTENDED fig-FORTH manual.) Then FLUSH the modified screens. Now load the screen in the usual way. A word followed by a "?" will appear if you edited the screen(s) incorrectly.

To reverse the process, and reinstate these words, just remove the parentheses and FLUSH the screens again.

CREATING AN AUTOMATICALLY LOADED PROGRAM

Once you've written a FORTH program, you may wish to set it up so that it can be loaded and run easily, even by someone who doesn't know FORTH. One nice way is to make your program autoloading using the following method:

1. Load your entire program (all of the words and variables) into computer memory. Now, suppose MYGAME is the word you execute to run the program. You would then type in the following (and press RETURN):

```
' MYGAME CFA ' ABORT 6 + !
```

(Make sure you type in the two apostrophes around MYGAME CFA. This line replaces a CR word in the ABORT code so that execution transfers to MYGAME upon use of the word COLD, which includes initially loading the program into memory or pressing the SYSTEM RESET button.)

2. Now insert a formatted diskette, type SAVE , and press the RETURN key. Note that once you do this, you can't use the diskette for anything else.

3. Now you have a diskette that will automatically load into memory and start when you turn on your computer. Also, pressing SYSTEM RESET will cause your program to restart automatically.

AUTHOR'S COMMENTS

I would like to extend my most sincere thanks to Pat Mullarky for helping me with this package (especially with methods used in the words FGR , SGR , SCR? , and also the autoload method described in the Advanced Technical Information section) and for writing his excellent version of fig-FORTH. Fun-FORTH may smooth out some of the rough spots in EXTENDED fig-FORTH, but there weren't many to begin with, thanks to Pat.

I would also like to thank my father, David Gluck, M.D., for his help and endless encouragement throughout the whole process.

I encourage you to send me news of any bugs you find in fun-FORTH (along with some solutions, if possible). And, in the hope that I get around to writing fun-FORTH II, I would appreciate any suggestions you have for words to include in the package along the lines of player/ missile graphics, character set redefinition, and other techniques, plus non-graphics vocabulary, such as string-handling words.

Good luck, be FORTH-ful, and program!

APPENDIX: QUADRASKETCH

OVERVIEW

QuadraSketch is a small drawing program that demonstrates the use of various words in fun-FORTH. The program lets you paint colorful, symmetrical patterns in graphics modes 3, 5, and 7, and you can change any of the colors at any time.

You need a Joystick Controller to use QuadraSketch.

GETTING STARTED

Loading QuadraSketch

To load QuadraSketch, first load the following screens from EXTENDED fig-FORTH and fun-FORTH, using the method described under "Getting Started" at the start of the manual. (Note. You must load EXTENDED fig-FORTH to perform the following steps.)

1. From the fig-FORTH diskette:

```
39 LOAD   50 LOAD
```

2. From the fun-FORTH diskette:

```
1 LOAD   2 LOAD   4 LOAD
20 21 LOADS   24 LOAD
30 31 LOADS
```

3. Then load all the screens necessary for QuadraSketch by typing

```
64 LOAD
```

with the fun-FORTH disk inserted in the disk drive.

4. To run QuadraSketch, type

```
QSK
```

and press RETURN.

5. Plug a Joystick Controller into the leftmost controller jack at the front of the computer.

THE FIRST DISPLAY SCREEN

After the screen clears, this prompt displays:

```
Welcome to QuadraSketch.

Mode?
```

Choice of mode

Type the number of the graphics mode you want to draw in--3, 5, or 7 (pressing the RETURN key isn't necessary). If you type any other character, the program chooses a mode for you.

The screen then clears and a blinking gold dot displays in the center.

USING QUADRASKETCH

"Painting"

Pushing on the joystick, move the blinking dot diagonally upward and to the right. Notice that 3 other dots appear and move in opposite directions. One goes up and left, another down and left, and the third moves down and to the right.

As the "brushes" move, each leaves a trail behind it. This is how you paint with QuadraSketch. Also, notice that the brushes start out moving slowly and gradually speed up. This feature gives you more control to do "touch-ups" but also lets you paint large patterns quickly.

Locating your position

When you stop painting, the brush you were controlling begins to blink, letting you know where you are. When you stand still, you hear no sounds. But when you move, you hear a clicking noise that speeds up into a low buzz. If you try to go off the screen, a BOOM! alerts you as you hit the edge. (If you don't hear these sounds, turn up the volume on your television set.)

Erasing or moving without painting

To erase what you've drawn, or to move the brushes without leaving a trail of paint, hold down the red button on the joystick as you move. The sound changes to something like the "hyperwarp" noise from Star Raiders (tm).

Using more than one color

Up until now, you've been drawing with color 1 and erasing by using the background color. However, you have two more colors to draw with: colors 2 and 3. To use these, type the number of the color you wish to draw with (for example, 3). You'll hear a weird, cyclic noise. Now push the red button. The noise stops and you re-enter normal drawing mode. You may now draw with color 3.

The preset colors are as follows:

```
1  gold  (initial brush)
2  bright green
3  blue
4  black  (background; not a selectable paint color)
```

To change any of these colors (for example, to change color 1 from gold to purple), type the number of the color you want to change. Next, move the joystick left or right. The color you have selected will cycle up or down the 128-color scale (starting with the original color). If it reaches either end of the scale (black or bright yellow) it automatically wraps around. When you reach the color you want, stop moving the joystick and push the red button. You then return to normal drawing mode, with your brush color set to the color you changed (unless you changed color 4, the background, in which case your brush color stays what it was before your change).

Avoid changing a color before drawing with it on the screen. Unless you have a color sample on the screen, when you change the color, you'll think nothing is happening because you'll see no visible change.

RESTARTING QUADRASKETCH

To leave QuadraSketch, press the ESC key while in normal drawing mode. You can't leave the program while in the color-changing mode. After the screen clears, the prompt:

```
Finished? (Y/N)
```

displays. Type "N" to remain in QuadraSketch (the "Mode?" prompt then redisplays). Type "Y" to return to the standard FORTH prompt "ok".

ADVANCED TECHNICAL INFORMATION

The following is a screen-by-screen description of the words making up QuadraSketch.

SCREEN #64

```
SCR # 64
( QuadraSketch  QSK   jgJUN82 )

( if not already LOADed: )
( 1 LOAD 2 LOAD 4 LOAD )
( 20 21 LOADS 24 LOAD )
( 30 31 LOADS )

  9 VARIABLE XX 9 VARIABLE YY
  38 VARIABLE XL 22 VARIABLE YL
  3 VARIABLE GMD 1 VARIABLE PCOL
  0 VARIABLE DELC
  -->
```

Screen #64 contains the "variables" used in QuadraSketch. XX and YY contain the x and y coordinates of the blinking brush. XL and YL contain the x and y limits, which depend on the selected mode. GMD contains the selected graphics mode number. PCOL contains the current color number. DELC stores the current delay used when moving the brushes.

```
SCR # 65
( QSK page 2          jgJUN82 )

: DONE?  0 FGR CR CR
  ." Finished? (Y/N) " Y/N ;
( -- 1/0 )

: COLSWITCH  200 BELL 4 SFXB1
  171 172 SFXB2 48 - DUP
  708 + DUP 712 < IF 1 - ENDIF
  BEGIN 0 XSTK -DUP IF ASOFF
  SWAP DUP C@ ROT + SWAP DUP
  ROT SWAP C! 50 DELAY 4 SFXB1
  ENDIF 0 STRIG 0= UNTIL ASOFF
  DROP DUP 4 < IF PCOL ! ELSE
  DROP ENDIF SINIT 100 BELL ;
( ascol -- ) -->
```

DONE?   -- 1/0

This word, which is essentially the last word to be executed, asks if you're finished, and returns the appropriate flag.

Comment. DONE? uses the word Y/N, on screen #4.

COLSWITCH   ascol --

This word receives the ASCII code of the color number (for example, 49 instead of 1) from KINTERP (see below). It then lets you change the desired color as described earlier.

Comment. COLSWITCH uses many fun-FORTH words, including SFXB1 and SFXB2 , which produce the "weird noise" described earlier in this section; BELL , which sounds upon entering and exiting this word; XSTK , which returns values for the right or left movements of the joystick; and ASOFF , which shuts off all the sound.

SCREEN #66

```
SCR # 66
( QSK page 3            jgJUN82 )

: KINTERP  KEY DUP DUP 48 >
  SWAP 53 < AND IF COLSWITCH 0
  ELSE 27 = ENDIF ;
( -- 1/0 )

: BLINK  0 SOFF 0 XX @ YY @ PLOT
  10 DELAY PCOL @ XX @
  YY @ PLOT 100 DELC ! ;
( -- )

: DSND  0 STRIG IF SPOP ELSE
  0 DELC @ 2 / 8 4 SND ENDIF
  DELC @ DUP DELAY 5 - DELC ! ;
( -- ) -->
```

KINTERP -- 1/0

Executed upon the press of a key, this word either enters the color-changing mode (COLSWITCH) or sets the flag for exit from QuadraSketch.

BLINK no parameters

Executed if the brushes are not moving, this word blinks the direct-control brush, and sets the movement delay to its slowest.

DSND no parameters

This word creates the noises heard during movement and decrements and executes the delay (which controls the speed of the brush).

SCREEN #67

```
SCR # 67
( QSK page 4          jgJUN82 )

: DIR+   0 XSTK DUP XX @ + 0
  YSTK DUP YY @ + DUP 0 <
  SWAP YL @ > OR ROT DUP 0 <
  SWAP XL @ > OR OR IF 2DROP
  30 20 EXPLODE ELSE YY +!
  XX +! ENDIF ;
( -- )

: PLOT4 XX @ YY @ CPLOT
  XL @ XX @ - YY @ CPLOT
  XX @ YL @ YY @ - CPLOT
  XL @ XX @ - YL @ YY @ -
  CPLOT ;
( -- ) -->
```

DIR+ no parameters

This word changes the x and y coordinates of the brush based on the direction of the joystick. If the resulting coordinates are off-screen, it causes an audible explosion and does not change the coordinates.

Comment. DIR+ makes use of XSTK , YSTK , and EXPLODE from fun-FORTH.

PLOT4 no parameters

This word plots all four points using the color stored in COL.

Comment. CPLOT, from fun-FORTH, is used to perform each plot. COL is a variable from fun-FORTH screen #31.

SCREEN #68

```
SCR # 68
( QSK page 5            jgJUN82 )

: COLT  PCOL @ 0 STRIG * TCOL ;
( -- )

: COLH  PCOL @ TCOL ;
( -- )

: QDRAW  0 STICK 15 - IF COLT
  PLOT4 DIR+ COLH PLOT4 DSND
  ELSE BLINK ENDIF ;
( -- ) -->




```

COLT   no parameters

Sets the color for the next PLOT4 to either the current PCOL (for painting) or the background (for erasing), based on the joystick trigger.

<u>Comment</u>. Uses STRIG and TCOL from fun-FORTH.

COLH   no parameters

Sets the color for the next PLOT4 to the current PCOL.

QDRAW   no parameters

If the joystick is being moved, this word performs a single paint-and-move or a blink using previously defined words.

SCREEN #69

```
SCR # 69
( QSK page 6             jgJUN82 )

: QSETUP GMD @ FGR 1 TCOL XL @
  2 / DUP XX ! YL @ 2 / DUP
  YY ! CPLOT 1 PCOL ! ;
( -- )

: MODE? 0 FGR CR CR
  ." Welcome to QuadraSketch."
  CR CR ."  Mode? " KEY 48 - 3
  MAX 7 MIN DUP DUP 4 = SWAP 6
  = OR IF DROP 5 ENDIF DUP GMD
  ! DUP 3 = IF 38 XL ! 22 YL
  ! ENDIF DUP 5 = IF 78 XL !
  46 YL ! ENDIF 7 = IF 158 XL
  ! 94 YL ! ENDIF ; -->
```

QSETUP   no parameters

Sets up the screen for drawing by going into the selected mode, plotting the brush, and setting the default color to 1.

MODE?   no parameters

Prints the initial welcome and prompt and inputs the desired mode. It then finds the correct screen limits for that mode.

SCREEN #70

```
SCR # 70
( QSK page 7          jgJUN82 )

: QSK BEGIN SINIT MODE?
      BEGIN QSETUP
       BEGIN KEYINIT
        BEGIN QDRAW KEYBD?
        UNTIL KINTERP
       UNTIL 1
      UNTIL DONE?
     UNTIL ;

  ;S




```

QSK   no parameters

This is the word that puts it all together. When executed, it runs QuadraSketch.

Comment. The outer BEGIN UNTIL loop lets you restart based on your response to DONE?. The next loop sets QSETUP apart for execution. The next loop surrounds the drawing loop so that KINTERP can re-enter normal drawing or exit. The innermost loop performs the actual drawing until a key is pressed.

INDEX TO WORDS

The following index contains each fun-FORTH word followed by it stack diagram ("--" by itself means "no parameters") and the screen where it is located.

APX ATARI® PROGRAM EXCHANGE
P.O. Box 3705
Santa Clara, CA 95055

Review Form

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many of our authors are eager to improve their programs if they know what you want. And, of course, we want to know about any bugs that slipped by us, so that the author can fix them. We also want to know whether our instructions are meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program.

____

____

2. If you have problems using the program, please describe them here.

____

____

____

3. What do you especially like about this program?

____

____

____

4. What do you think the program's weaknesses are?

____

____

____

5. How can the catalog description be more accurate or comprehensive?

____

____

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program:

________ Easy to use
________ User-oriented (e.g., menus, prompts, clear language)
________ Enjoyable
________ Self-instructive
________ Useful (non-game programs)
________ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

____

____

____

8. What did you especially like about the user instructions?

____

____

____

9. What revisions or additions would improve these instructions?

____

____

____

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

____

____

11. Other comments about the program or user instructions:

____

____

____

From

____

____

____

STAMP

ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

[seal here]