FORTH GRAPHICS TOOLBOX (A USER'S GUIDE FOR USE WITH RFF FORTH)

BY HANSEOK KO
UNDERWATER SYSTEMS DEPARTMENT

14 JUNE 1991

Approved for public release; distribution is unlimited.

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 e Silver Spring, Maryland 20910-5000

91 1125 100

91-16493
FORTH GRAPHICS TOOLBOX
(A USER’S GUIDE FOR USE WITH RFF FORTH)

BY HANSEOK KO
UNDERWATER SYSTEMS DEPARTMENT

14 JUNE 1991

Approved for public release; distribution is unlimited.

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000
FOREWORD

The FORTH GRAPHICS TOOLBOX is to be used to develop FORTH based application software. This document is intended to provide a manual detailing the procedures and usage.

This document has been reviewed by the users in the Simulation and Training Section for its technical integrity and the Underwater Signal Processing Branch's line management for elements of format and style.

The GRAPHICS TOOLBOX has evolved over a period of two years with input from many users. The author would like to extend his thanks to several people for their input. Kit Yan is credited with the development of many graphics routines at the beginning of this project; the entire project was made much easier because of the strong foundation laid out initially by Yan. Bob Davis, Phil Craun, and Paul Craun provided much useful technical insight in the development of this document. Finally, Ira Rosenbaum, Mark Williams, Bob Otte, and John Sherman provided the encouragement to the author in pursuing this project.

If you have questions or comments about the TOOLBOX, please contact U25 (Hanseok Ko), (301)394-2372. Comments are welcome and will be considered when the TOOLBOX is revised.

Approved by:

C. Kalivretenos, Deputy Department Head
Underwater Systems Department
# CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2 INSTALLATION</td>
<td>2-1</td>
</tr>
<tr>
<td>3 HOW TO READ TOOLBOX</td>
<td>3-1</td>
</tr>
<tr>
<td>4 FORTH GRAPHICS TOOLBOX</td>
<td>4-1</td>
</tr>
<tr>
<td>5 REFERENCES</td>
<td>5-1</td>
</tr>
</tbody>
</table>

## Appendixes

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A FUNCTIONAL DESCRIPTION</td>
<td>A-1</td>
</tr>
<tr>
<td>B SOURCE CODE</td>
<td>B-1</td>
</tr>
</tbody>
</table>
ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>FORTH STACK DIAGRAM</td>
<td>3-2</td>
</tr>
</tbody>
</table>

TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>VIDEO ENVIRONMENT SETUP ROUTINES</td>
<td>4-1</td>
</tr>
<tr>
<td>4-2</td>
<td>DIRECT VIDEO DRAWING COMMANDS</td>
<td>4-2</td>
</tr>
<tr>
<td>4-3</td>
<td>BIOS CALLED VIDEO DRAWING COMMANDS</td>
<td>4-2</td>
</tr>
<tr>
<td>4-4</td>
<td>DIRECT VIDEO DRAWING APPLICATION COMMANDS</td>
<td>4-2</td>
</tr>
<tr>
<td>4-5</td>
<td>BIOS CALLED VIDEO DRAWING APPLICATION COMMANDS</td>
<td>4-3</td>
</tr>
<tr>
<td>4-6</td>
<td>DIRECT VIDEO DEMO ROUTINES</td>
<td>4-3</td>
</tr>
<tr>
<td>4-7</td>
<td>BIOS CALLED VIDEO DEMO ROUTINES</td>
<td>4-3</td>
</tr>
<tr>
<td>4-8</td>
<td>BIOS FUNCTION CALLS</td>
<td>4-4</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

FORTH GRAPHICS TOOLBOX is a rich collection of graphics routines immediately useful for all FORTH-based application software running on IBM-PC clone microcomputers. The routines are built based on graphics related primitives of both video BIOS call functions and Direct-video functions. The user can develop more exotic application software based on the routines listed in this package.

The central features of the FORTH GRAPHICS TOOLBOX are functions for:

- Video screen environment setup.
- Direct and BIOS called video drawing of a point, a line, and a circle.
- Direct and BIOS called drawing application.
- Demonstration of the package's graphics capabilities.

These functions are implemented in the FORTH environment under the file name GRAPHICS.SEQ. Accessing this file will allow the user to make changes, add features, or learn how a given algorithm works. New application routines can be developed easily by first loading GRAPHICS.SEQ and experimenting with the application SEQ-files.

The GRAPHICS TOOLBOX has evolved over a period of 2 years with input from many users, including those mentioned in the Foreword.
CHAPTER 2
INSTALLATION

FORTH GRAPHICS TOOLBOX can be installed by first getting into RFF FORTH working space. RFF\(^1\) is a 16-bit FORTH system built upon the work of several people: the original F83 system by Laxen and Perry, the "FF" system by Tom Zimmer et. al., and numerous contributions of Robert Davis. FORTH\(^2\) is a language that begins with a powerful set of standard commands, then provides the mechanism by which a user can define his/her own commands. The structured process of building definitions upon previous definitions is the FORTH equivalent of high-level coding. Alternatively, words may be defined directly in assembler mnemonics, using FORTH's assembler. All commands are interpreted by the same interpreter and compiled by the same compiler. The user can get into the RFF FORTH environment by typing 'RFF' from a directory containing the 'RFF.EXE' file.

Once in the RFF FORTH environment, type:

```
FLOAD GRAPHICS
```

to load GRAPHICS.SEQ. After GRAPHICS.SEQ is loaded, test it by invoking a demonstration graphics routine such as "TELLIPSE." Upon invoking "TELLIPSE", the user should see random sets of different colored concentric rings displayed on the screen.
CHAPTER 3
HOW TO READ THE TOOLBOX

Each routine or function, presented in Reference A, is tagged as either CODE or WORD. CODE implies that it is an assembly routine that exists because it is either a BIOS called function or an attempt to save processing time. WORD implies that it is colon (:) defined and structured in order to get the full advantage of the FORTH environment. "Category" is listed to provide a quick reference to the routine's background such as whether it is a direct video or BIOS called function.

A stack diagram is provided adjacent to the name of each routine in the parenthesis. For example, the first line for routine LINE looks like this:

\[
\text{LINE} \quad (x_1 \ y_1 \ x_2 \ y_2\ \text{color} \ # \ -\ -)
\]

The order of inputs typed onto the screen is important since it determines the inputs' respective positions on the stack. In the above case, the computer performs the operation in accordance to the task defined by LINE by either pushing or popping the numbers on the stack. The FORTH's stack is described as "last-in, first-out" (LIFO). This means that the only accessible value at any given time is the top value. The system reads input from left to right and executes each word in turn. For input, the rightmost value on the screen will end up on top of the stack. For output, the rightmost value on the screen came from the lowest position on the non-empty column of the stack. The order of inputs with respect to the top of the stack, for the LINE routine, is as follows:
If a numerical output is desired, the corresponding output variables are listed to the right of the dash (--) in the stack diagram. But if no output variable is listed, as in the case of LINE routine, then an action on the hardware such as "drawing" is expected as the output.

The ranges of the legitimate numerical values are indicated in the Description block. Most values must be given as integers; however, some routines require real numbers as input. When real numbers are required, the input variables in the stack diagram will be denoted by a decimal point as shown below.

\[
\text{AUTOSCALE} \quad (x_1, x_2, x, y, \text{color}, \text{npt}, \text{hv} -- )
\]

In general, there are two categories of graphic routines: the direct-video and the BIOS call. The BIOS call routines are denoted by a "-_BIOS" postfix attached to the syntax of

\[3-2\]
the direct-video counterparts. For example, direct-video's AST routine has BIOS call counterpart AST_BIOS which is invoked with software interrupt 10H.

The direct-video routines are at least 10 times faster than the BIOS call routines in getting the corresponding image on the screen. However, the BIOS call routines may become handy if there is a mismatch between the direct-video routines and the display mode type. For example, a program using only BIOS function calls for video output will run in almost any MS-DOS environment, regardless of the video hardware, including (but not limited to) the entire IBM PC and PS/2 family.

The available colors are simple combinations of the primary colors red, green, and blue mapped into 16 colors as follows:

0 = Black
1 = Blue
2 = Green
3 = Cyan
4 = Red
5 = Violet
6 = Yellow (brown)
7 = White
8 = Black (gray)
9 = Intense blue
10 = Intense green
11 = Intense cyan
12 = Intense red
13 = Intense violet
14 = Intense yellow
15 = Intense white
CHAPTER 4
FORTH GRAPHICS TOOLBOX
REFERENCE

This chapter contains a listing of all FORTH Graphics Toolbox routines grouped by subject, listed in alphabetical order, and followed by a brief description of the routine.

<table>
<thead>
<tr>
<th>TABLE 4-1. VIDEO SCREEN ENVIRONMENT SETUP ROUTINES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VIDEO SCREEN ENVIRONMENT SETUP ROUTINES</strong></td>
</tr>
<tr>
<td>AND__VIDEO</td>
</tr>
<tr>
<td>CGA__HI</td>
</tr>
<tr>
<td>CO80</td>
</tr>
<tr>
<td>EGA__HI</td>
</tr>
<tr>
<td>EGA__LO</td>
</tr>
<tr>
<td>8x8FONT</td>
</tr>
<tr>
<td>8x14FONT</td>
</tr>
<tr>
<td>8x16FONT</td>
</tr>
<tr>
<td>NORMAL__VIDEO</td>
</tr>
<tr>
<td>OR__VIDEO</td>
</tr>
<tr>
<td>SET__GRAPHMODE</td>
</tr>
<tr>
<td>XOR__VIDEO</td>
</tr>
<tr>
<td>VGA__HI</td>
</tr>
</tbody>
</table>
### TABLE 4-2. DIRECT VIDEO DRAWING COMMANDS

<table>
<thead>
<tr>
<th>DIRECT VIDEO DRAWING COMMANDS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND_VIDEO</td>
<td>Latched pixels ADed</td>
</tr>
<tr>
<td>ELLIPSE</td>
<td>Draws an ellipse/circle</td>
</tr>
<tr>
<td>HORIZ_LINE</td>
<td>Draws an horizontal straight line</td>
</tr>
<tr>
<td>LINE</td>
<td>Draws a straight line of any orientation</td>
</tr>
<tr>
<td>VERT_LINE</td>
<td>Draws a vertical straight line</td>
</tr>
<tr>
<td>CHI: PLOT</td>
<td>Plot a character</td>
</tr>
</tbody>
</table>

### TABLE 4-3. BIOS CALLED VIDEO DRAWING COMMANDS

<table>
<thead>
<tr>
<th>BIOS CALLED VIDEO DRAWING COMMANDS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELLIPSE_BIOS</td>
<td>Draws an ellipse/circle</td>
</tr>
<tr>
<td>HORIZ_LINE_BIOS</td>
<td>Draws an horizontal straight line</td>
</tr>
<tr>
<td>LINE_BIOS</td>
<td>Draws a straight line of any orientation</td>
</tr>
<tr>
<td>PUT_PIXEL</td>
<td>Plot a point</td>
</tr>
<tr>
<td>VERT_LINE_BIOS</td>
<td>Draws a vertical straight line</td>
</tr>
<tr>
<td>CHRPLOT_BIOS</td>
<td>Plot a character</td>
</tr>
</tbody>
</table>

### TABLE 4-4. DIRECT VIDEO DRAWING APPLICATION COMMANDS

<table>
<thead>
<tr>
<th>DIRECT VIDEO DRAWING APPLICATION COMMANDS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSCALE</td>
<td>Draws a scale with tick marks</td>
</tr>
<tr>
<td>OUTTEXTXY</td>
<td>Plot a string of characters</td>
</tr>
<tr>
<td>SIGPLOTB</td>
<td>Plot a string of characters</td>
</tr>
</tbody>
</table>
### TABLE 4-5. BIOS CALLED VIDEO DRAWING APPLICATION COMMANDS

<table>
<thead>
<tr>
<th>BIOS CALLED VIDEO DRAWING APPLICATION COMMANDS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOSCALE_BIOS</td>
<td>Draws a scale with tick marks</td>
</tr>
<tr>
<td>OUTTEXTXY_BIOS</td>
<td>Plot a string of characters</td>
</tr>
</tbody>
</table>

### TABLE 4-6. DIRECT VIDEO DEMO ROUTINES

<table>
<thead>
<tr>
<th>DIRECT VIDEO DEMO ROUTINES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERICA</td>
<td>Writes texts in graphics mode</td>
</tr>
<tr>
<td>AST</td>
<td>Draws a set of scales with tick marks</td>
</tr>
<tr>
<td>FIREWORK1</td>
<td>Draws random flashes of fireballs</td>
</tr>
<tr>
<td>FIREWORK3</td>
<td>Draws random flashes of fireballs</td>
</tr>
<tr>
<td>TELLIPSE</td>
<td>Draws random sets of rings</td>
</tr>
<tr>
<td>TLIN2</td>
<td>Draws random sets of lines</td>
</tr>
<tr>
<td>TXT</td>
<td>Writes texts in four orientations</td>
</tr>
</tbody>
</table>

### TABLE 4-7. BIOS CALLED VIDEO DEMO ROUTINES

<table>
<thead>
<tr>
<th>BIOS CALLED VIDEO DEMO ROUTINES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERICA_BIOS</td>
<td>Writes texts in graphics mode</td>
</tr>
<tr>
<td>AST_BIOS</td>
<td>Draws a set of scales with tick marks</td>
</tr>
<tr>
<td>FIREWORK2</td>
<td>Draws random flashes of fireballs</td>
</tr>
<tr>
<td>TLINE_BIOS</td>
<td>Draws random sets of lines</td>
</tr>
<tr>
<td>TXT_BIOS</td>
<td>Writes texts in four orientations</td>
</tr>
<tr>
<td>BIOS FUNCTION CALLS</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>FONTAD</td>
<td>Get FONT address within EGA/VGA</td>
</tr>
<tr>
<td>GET_GRAPHMODE</td>
<td>Get info about current graphic mode</td>
</tr>
<tr>
<td>GET_PIXEL</td>
<td>Read color info at (x,y) pixel location</td>
</tr>
<tr>
<td>GET_XY</td>
<td>Read current cursor position</td>
</tr>
<tr>
<td>GOTO_XY</td>
<td>Move cursor to (x,y) position</td>
</tr>
<tr>
<td>PUT_PIXEL</td>
<td>Plot a point at (x,y) pixel location</td>
</tr>
<tr>
<td>READ_CHAR</td>
<td>Read character at current cursor position</td>
</tr>
<tr>
<td>SCROLL_PAGEDOWN</td>
<td>Scroll texts top to bottom on screen</td>
</tr>
<tr>
<td>SCROLL_PAGEUP</td>
<td>Scroll texts bottom to top on screen</td>
</tr>
<tr>
<td>SET_ACTIVEPAGE</td>
<td>Select a page as active page for graphics</td>
</tr>
<tr>
<td>SET_BORDER</td>
<td>Draw a border line</td>
</tr>
<tr>
<td>SET_GRAPHMODE</td>
<td>Set the screen to various graphics modes</td>
</tr>
<tr>
<td>SET_PALETTE</td>
<td>Change palette's color entry to new color</td>
</tr>
<tr>
<td>WRITE_CHAR</td>
<td>Put character at current cursor position</td>
</tr>
<tr>
<td>WRITE_TCHAR</td>
<td>Put character at current cursor position and move cursor to next position</td>
</tr>
</tbody>
</table>
REFERENCES


AMERICA (x y ...)  

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:
- Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.) is invoked and given (x,y) coordinates, AMERICA writes text "UNITED STATES OF AMERICA" in four 90-deg orientations centered at (X,Y).

Examples:

VGA_HI 300 200 AMERICA

See also:
- AMERICA_BIOS

AMERICA_BIOS (x y ...)

Type: WORD

Category: BIOS call video graphic drawing routine

Purpose:
- Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.) is invoked and given (x,y) coordinates, AMERICA_BIOS writes text "UNITED STATES OF AMERICA" in four 90-deg orientations centered at (X,Y).

Examples:

VGA_HI 300 200 AMERICA_BIOS

See also:
- AMERICA
AND_VIDEO ( ... )

Type: WORD
Category: Video environment controlling routine
Purpose:
   Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.
Description:
   AND_VIDEO specifies the Data Rotate/Function Select register's (D3H) 2-bit fields to 00001000. This bit pattern forces latched pixels to be ANDed when updated. Note that the variable PIXEL_MODE stores the 2-bit fields.

Examples:
   VGA_H1 AND_VIDEO 250 300 12 PUT_PIXEGA

See also:
   OR_VIDEO, XOR_VIDEO

AST ( x, y, ... )

Type: WORD
Category: Direct-video graphic drawing routine
Purpose:
   Demonstration of graphics drawing capability.
Description:
   After GRAPHICS MODE (VGA_H1, EGA_H1, CGA_LO, etc.) is invoked and given the two boundaries (xl, x2) with decimal numbers, AST will draw a set of differently scaled axes.

Examples:
   VGA_H1 20.0 2000.0 AST

See also:
   AST_BIOS
AST_BIOS (x, y, ...)  

Type: WORD  

Category: BIOS call graphic drawing routine  

Purpose:  
Demonstration of graphics drawing capability.  

Description:  
After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.) is invoked and given the two boundaries (x1, x2) with decimal numbers, AST_BIOS will draw a set of differently scaled axes.  

Examples:  
VGA_HI 20.0 2000.0 AST_BIOS  

See also:  
AST  

AUTOSCALE (x1, x2, y color rpt hv --)  

Type: WORD  

Category: Direct graphics routine  

Purpose:  
Draw scales with tick marks  

Description:  
Given two real numbers x1 and x2, AUTOSCALE draws a straight line between the two numbers with tick marks and labels whose sizes are automatically adjusted to minimize cluttering.  

AUTOSCALE requires the following inputs:  
x1 == lower limit of two real numbers  
x2 == upper limit of two real numbers  
x == x-coordinate of the line's starting point  
y == y-coordinate of the line's starting point  
color == color number (0, 15)  
rpt == length of the line in terms of the number of pixels (i.e., (1,640) for horizontal orientation and (1,480) for vertical orientation)  
hv == 1 for horizontal  
0 for vertical  

Examples:  
0. 1000.0 20 250 12 300 1 AUTOSCALE  

See also:  
AUTO_RANGE, AUTOSCALE_BIOS
AUTOSCALE_BIOS (x1, x2, x y color npt hv --)

Type: WORD

Category: Video BIOS call graphics routine

Purpose: Draw a scale with tick marks

Description: Given two real numbers, x1 and x2, AUTOSCALE_BIOS draws a straight line between the two numbers with tick marks and labels whose sizes are automatically adjusted to minimize cluttering.

AUTOSCALE_BIOS requires the following inputs:

x1 == lower limit of two real numbers
x2 == upper limit of two real numbers
x == x-coordinate of the line's starting point
y == y-coordinate of the line's starting point
color == color number (0, 15)
npt == length of the line in the number of pixels (i.e., (1,640) for horizontal orientation and (1,480) for vertical orientation)
hv == 1 for horizontal
      0 for vertical

Examples:
0.1000.0 20 250 12 300 1 AUTOSCALE_BIOS

See also: AUTO_RANGE, AUTOSCALE

CGA_HI (--) Type: WORD

Category: Video screen environment setup routine

Purpose: Set the system to the specified graphics mode

Description: Invoking CGA_HI will bring the screen to high-resolution-CGA mode by calling SET GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x200 2-color graphics: one must be black.

Examples: CGA_HI 50 50 200 200 12 LINE_BIOS

See also: CGA_HI, EGA_HI, EGA_LO, VGA_HI, VGA_LO
CHNPLOT BIOS

( char color# x y limh limv hv -- )

Type: WORD

Category: Video BIOS call character drawing routine

Purpose:

Allow characters to be drawn in graphics mode on graphics coordinates.

Description:

Upon invoking CHNPLOT BIOS during graphics mode, a character will be drawn in pixel coordinates (x,y) in either horizontal or vertical orientation.

Input requirement:

charr == IBM PC character set coded in decimal

A=65
B=66
C=67
D=68
E=69
F=70
G=71
H=72
I=73
J=74
K=75
L=76
M=77
N=78
O=79
P=80
Q=81
R=82
S=83
T=84
U=85
V=86
W=87
X=88
Y=89
Z=90
a=97
b=98
c=99
d=100

Color == color# (0,15)

(See "How to read the Toolbox" for color codes)

x == character position's x-coordinate (0,639) for VGA_HI

y == character position's y-coordinate (0,479) for VGA_HI

limh == bottom window clip == 0

limv == top window clip (i.e., 639 for horizontal orientation and 479 for vertical orientation)

hv == 0 vertical

1 horizontal

Examples:

VGA_HI 8x14FONT 65 14 10 20 0 639 1 CHNPLOT

See also:

CHNPLOT BIOS

---------------------

CHNPLOT

( char color# x y limh )

Type: WORD

Category: Direct video character drawing routine

Purpose:

Allow characters to be drawn in graphics mode on graphics coordinates.

Description:

Upon invoking CHNPLOT during graphics mode, a character will be drawn in pixel coordinates (x,y) in either horizontal or vertical orientation.

Input requirement:

charr == IBM PC character set coded in decimal

A=65
B=66
C=67
D=68
E=69
F=70
G=71
H=72
I=73
J=74
K=75
L=76
M=77
N=78
O=79
P=80
Q=81
R=82
S=83
T=84
U=85
V=86
W=87
X=88
Y=89
Z=90
a=97
b=98
c=99
d=100

Color == color# (0,15)

(See "How to read the Toolbox" for color codes)

x == character position's x-coordinate (0,639) for VGA_HI

y == character position's y-coordinate (0,479) for VGA_HI

limh == bottom window clip == 0

limv == top window clip (i.e., 639 for horizontal orientation and 479 for vertical orientation)

hv == 0 vertical

1 horizontal

Examples:

VGA_HI 8x14FONT 65 14 10 20 0 639 1 CHNPLOT

See also:

CHNPLOT
COBO  (  ..  )

Type: WORD

Category: Video screen environment setup routine

Purpose:
Switch from screen's graphic mode to text mode.

Description:
Invoking COBO will bring the screen to 80 column alphanumeric (e.g., text) mode by calling SET_GRAPHMODE, a video BIOS function routine.

During the process, the screen gets cleared.

Examples:
COBO

See also:
CGA_HI, EGA_HI, EGA_LO, VGA_HI, VGA_LO

---

EGA_HI  (  ..  )

Type: WORD

Category: Video screen environment setup routine

Purpose:
Set the system to the specified graphics mode.

Description:
Invoking EGA_HI will bring the screen to high-resolution-EGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x350 16-color graphics

It also sets up the graphics character heights to be 14 units (6 pixels) high. (VGA_HI has 16 units high character size)

During the process, the screen gets cleared.

Examples:
EGA_HI 50 50 200 200 12 LINE_BIOS

See also:
CGA_HI, CGA_LO, EGA_LO, VGA_HI, VGA_LO
Type: WORD

Category: Video screen environment setup routine

Purpose:
Set the system to the specified graphics mode

Description:
Invoking EGA_LO will bring the screen to low-resolution-EGA
mode by calling SET_GRAPHMODE, a video BIOS function routine.
This call corresponds to setting the video to:

640x200 16-color graphics

It also sets up the graphics character heights to be 14 units
high. (VGA_HI has 16 units high character size)

During the process, the screen gets cleared.

Examples.

EGA_LO 50 50 200 200 12 LINE_BIOS

See also:
CGA_HI, EGA_LO, VGA_HI, VGA_LO

EGAPIX_ADDR (x y --)

Type: LABEL

Category: Assembly code subroutine needed for direct-video
graphics routines

Purpose: Determine buffer address of pixel in native EGA/VGA modes:
320x200 16-color 640x200 16-color 640x250 16-color
640x350 monochrome (4-color) 640x480 2-color 640x480 16-color

Description:
This subroutine converts pixel coordinates to the corresponding
byte and bit offsets in the video buffer. In graphics modes, the
video buffer can be thought of as a flat, two-dimensional array of
pixels with its origin at the upper left corner. What is visible
on the screen is a subset of the pixels represented in the buffer.
On the EGA, the video buffer can contain only one screenful of
pixels, so the first byte in the buffer represents the pixels in
the screen's upper left corner. On the EGA, MC68, and VGA,
however, the video buffer can store several screenfuls of pixels.
You can thus select which portion of the video buffer appears on
the screen.

Every pixel on the screen can be identified by a unique pair of
(x,y) coordinates relative to the screen's upper left corner. Each
(x,y) pair also corresponds to a particular byte offset in the
video buffer and a bit offset in that byte. Thus, given a pixel's
(x,y) coordinates on the screen, you can compute where in the video
buffer the pixel is represented. Transforming pixel coordinates
to a buffer offset involves simple logic. Begin by calculating the
offset of the start of pixel row y. (For EGA and Hercules graphics
modes, this calculation accounts for the interleaving of the video
buffer.) To this value, add the byte offset of the xth pixel in
the row. Finally, add the byte offset of the start of the
displayed portion of the video buffer to obtain the final byte
offset of the pixel.

PixelByteOffset = RowOffset(y) + ByteOffset(x) + OriginOffset

The bit offset of the pixel within the byte that contains its value
depeads only on the number of pixels represented in each byte of
the video buffer. However, it is more practical to represent a
pixel's bit offset as a bit mask rather than as an ordinal bit
number. This is done easily with a logical shift instruction.

Examples:

CALL EGAPIX_ADDR

See also:
PUT_PIXEGA, GET_PIXEGA, VERT_LINE, HORIZ_LINE, LS_LINE, etc
for examples of usage.
ELLIPSE ( color# b a yc xc -- )

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide an ellipse/circle drawing capability

Description:

This routine draws an ellipse in EGA/VGA graphics modes with the following inputs:

- color# = (0,15) (SEE "How to read the toolbox" for color codes)
- b = minor axis,
- a = major axis,
- xc = x-coordinate of ellipse's center
- yc = y-coordinate of ellipse's center

Examples:

12 50 100 300 300 ELLIPSE to draw an ellipse
12 50 50 300 300 ELLIPSE to draw a circle

See also:

LINE, LINE BIOS

FIREWORK1 ( -- )

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA HI is invoked, FIREWORK1 will draw random flashes of fireballs resembling fireworks lighting up in the sky. The fireball's textures are made up of rays of different color.

Examples:

VGA HI FIREWORK1

See also:

FIREWORK2, FIREWORK3
FIREWORK2

Type: WORD

Category: BIOS call video graphic drawing routine

Purpose:
Demonstration of graphics drawing capability.

Description:
After VGA_HI is invoked, FIREWORK2 will draw random flashes of fireballs resembling fireworks lighting up the sky. The fireball's textures are made up of rays of same color.

Examples:
VGA_HI FIREWORK2

See also:
FIREWORK1, FIREWORK3

FIREWORK3

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:
Demonstration of graphics drawing capability.

Description:
After VGA_HI is invoked, FIREWORK3 will draw random flashes of fireballs resembling fireworks lighting up the sky. The fireball's textures are made up of rays of same color.

Examples:
VGA_HI FIREWORK3

See also:
FIREWORK1, FIREWORK2
**8x8FONT**

**Type:** WORD  
**Category:** General  
**Purpose:**
Allow the use of different size fonts when using CHRPLT (character plotting routine)

**Description:**
Invoking 8x8FONT returns the address of where the font character is and the height of each character.

- Input: none
- Output: FONZ == the address where the font is located  
  CHRHIGHT == the height (8 pixels) of the font in pixels

**Examples:**

VGA_H 8x8FONT 65 14 200 200 0 639 1 1CHRPLT

**See also:**

8x14FONT, 8x16FONT

---

**8x14FONT**

**Type:** WORD  
**Category:** General  
**Purpose:**
Allow the use of different size fonts when using CHRPLT (character plotting routine)

**Description:**
Invoking 8x14FONT returns the address of where the font character is and the height of each character.

- Input: none
- Output: FONZ == the address where the font is located  
  CHRHIGHT == the height (14 pixels) of the font in pixels

**Examples:**

VGA_H 8x14FONT 65 14 200 200 0 639 1 1CHRPLT

**See also:**

8x8FONT, 8x16FONT
**8x16FONT**  
( ... )

**Type:** WORD

**Category:** General

**Purpose:**
Allow the use of different size fonts when using  
CHRPLOT (character plotting routine)

**Description:**
Invoking 8x16FONT returns the address of where the  
font character is and the height of each character.

**Input:** none

**Output:** FONZ == the address where the font is located  
CHRMGET == the height (16 pixels) of the font in pixels

**Examples:**

```
VGA_HI 8x16FONT 65 14 200 200 0 639 1 CHRPLOT
```

**See also:**

8x8FONT, 8x14FONT

---

**FONTAD**  
( ... seg offset )

**Type:** CODE

**Category:** BIOS function call 17  
Subservice call 30

**Purpose:**
Get address of FONT located within the EGA/VGA BIOS

**Description:**
FONTAD is a BIOS function call #11H routine which allows to  
get the following current character generator information (i.e  
AL=30H).

**Possible Input:**
- BH = 2 Address of 8x14 character table  
- #3 Address of 8x7 character table  
- #4 Address of second half of 8x8 character table  
- #5 Address of 9x14 alternate character table  
- #6 Address of 8x16 character table  
- #7 Address of 9x16 alternate character table

**Returned values:**
- ES:BP = address of character definition table

**Examples:**

```
0200 FONTAD  
( returns location of FONT in SEGMENT and OFFSET as used by  
  8x16FONT or 8x14FONT )
```

**See also:**

8x16FONT, 8x14FONT, 8x8FONT
GET_GRAPHMODE 
( -- video mode )

Type: CODE
Category: BIOS function call 15
Purpose:
Get information about the current graphic mode
Description:
GET_GRAPHMODE is a BIOS function call routine which imports
the current graphic mode information.

Input: none
Possible Output:
0 & 1 - 40 column alphanumeric (CGA compatible)
2 & 3 - 80 column alphanumeric (CGA compatible)
4 & 5 - 320x200 4-color graphics limited to 2 palettes (CGALo)
6 - 640x200 2-color graphics: one must be black (EGAHi)
7 - monochrome alphanumeric (monochrom adapter compatible)
8 - 12 - reserved
13 - 320x200 16 color
14 - 640x200 16 color (EGALo)
15 - 640x350 monochrome graphics (EGAMonoHi)
16 - 640x350 16 color (EGAHi)
17 - 640x480 monochrome graphics (VGAMonoHi)
18 - 640x480 16 color (VGAMi)
19 - 320x200 256 color (VGA only)

Examples:
GET_GRAPHMODE

See also:
SET_GRAPHMODE

GET_PIXEGA 
( x y -- color )

Type: CODE
Category: Direct video
Purpose:
Get color information at the indicated pixel location
Description:
GET_PIXEGA is a direct video routine which imports
the color# (0,15) at (x,y) coordinate pixel position.

Input:
\[ x = x\text{-coordinate (0,639)} \]
\[ y = y\text{-coordinate (0,479)} \]
Output: color code numbers (0,15)
(See "How to read the toolbox" for color codes)

Examples:
VGA HI TELLIPSE  (draws many sets of ellipse rings)
40 50 GET_PIXEGA  (color# for pixel(40,50) is retrieved)

See also:
GET_PIXEL  (equivalent BIOS call routine)
PUT_PIXEGA
GET_PIXEL  ( x y -- color# )

Type: CODE

Category: Video BIOS function call 13

Purpose: Read color information at the indicated pixel location

Description: GET_PIXEL is a video BIOS call routine which imports the color# (0,15) at (x,y) coordinate pixel position.

Input:
  x == x-coordinate (0,639)
  y == y-coordinate (0,479)

Output: color code numbers (0,15)
  (See "How to read the toolbox" for color codes)

Examples:
  VGA HI TELLIPSE  (draws many sets of ellipse rings)
  40 50 GET_PIXEL  (color# for pixel(40,50) is retrieved)

See also:
  GET_PXREGA  (equivalent direct-video routine)
  PUT_PXREGA

GET_XY  ( -- x y )

Type: CODE

Category: Video BIOS function call 3

Purpose: Read current cursor position

Description: GET_XY is a video BIOS call routine which imports the (x,y) position coordinates of the cursor.

Input: none

Output:
  x == x-coordinate (0,79)
  y == y-coordinate (0,24)

Examples:
  GET_XY

See also:
  GOTO_XY
GOTO_XY  
( x y \ldots )

Type:  CODE

Category:  Video BIOS function call 2

Purpose:  
Move cursor position to the designated coordinates (x,y)

Description:
GOTO_XY is a video BIOS call routine which allows positioning the
cursor to the coordinates (x,y).

Input:
x = x-coordinate (0,79)
y = y-coordinate (0,24)

Output: none

Examples:
GOTO_XY

See also:
GET_XY

---

HORIZ_LINE  
( y x1 x2 color\# \ldots )

Type:  CODE

Category:  Direct-video drawing routine

Purpose:  
Provide a horizontal (slope=0) straight line drawing capability

Description:
This routine draws an horizontal line in EGA/VGA graphics
modes with the following inputs:
\( y \) = vertical position (i.e., (0,480) if VGA_HI)
x1 = start point in the x-axis (i(0,640) if VGA_HI)
x2 = end point in the x-axis (i(0,640) if VGA_HI)
color\# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:
VGA_HI 100 20 300 12 HORIZ_LINE

See also:
HORIZ_LINE_BIOS, VERT_LINE, LINE,
**HORIZ_LINE_BIOS**

(\texttt{y \times x1 \times x2 \text{ color#} \ldots})

**Type:** WORD

**Category:** Video BIOS call drawing routine

**Purpose:**

Provide a horizontal (i.e., slope=0) straight line drawing capability

**Description:**

This routine draws an horizontal line in EGA/VGA graphics modes with the following inputs:

- \texttt{y} = vertical position (i.e., \(0,480\) if VGA_HI)
- \texttt{x1} = start point in the \texttt{x-axis} (\(0,640\) if VGA_HI)
- \texttt{x2} = end point in the \texttt{x-axis} (\(0,640\) if VGA_HI)
- \texttt{color#} = line's color \((1,15)\)

(See "How to read the toolbox" for color codes)

**Examples:**

\texttt{VGA_HI 100 20 300 12 HORIZ_LINE_BIOS}

See also:

HORIZ_LINE, VERT_LINE, LINE, LINE_BIOS

---

**NS_LINE**

(\texttt{x1 \times y1 \times x2 \times y2 \text{ color#} \ldots})

**Type:** CODE

**Category:** Direct-video drawing routine

**Purpose:**

Provide a straight line (slope \(=1\)) drawing capability and supports \texttt{LINE} routine

**Description:**

This routine draws a sloped line in EGA/VGA graphics modes with the following inputs:

- \texttt{x1} = start point in the \texttt{x-axis} (\(0,640\) if VGA_HI)
- \texttt{y1} = start point in the \texttt{y-axis} (i.e., \(0,480\) if VGA_HI)
- \texttt{x2} = end point in the \texttt{x-axis} (\(0,640\) if VGA_HI)
- \texttt{y2} = end point in the \texttt{y-axis} (i.e., \(0,480\) if VGA_HI)
- \texttt{color#} = line's color \((1,15)\)

(See "How to read the toolbox" for color codes)

**Examples:**

\texttt{VGA_HI 100 20 300 12 NS_LINE}

See also:

LINE, LS_LINE, HORIZ_LINE_BIOS, VERT_LINE, LINE,
**LINE** 
(x1 y1 x2 y2 color# ...)

**Type:** CODE

**Category:** Direct-video drawing routine

**Purpose:**
Provide the capability of drawing a straight line in any orientation.

**Description:**
This routine draws a straight line in EGA/VGA graphics modes with the following inputs:

- x1: start point in the x-coordinate (0,640) if VGA_Hi
- y1: start point in the y-coordinate (0,480) if VGA_Hi
- x2: end point in the x-coordinate (0,640) if VGA_Hi
- y2: end point in the y-coordinate (0,480) if VGA_Hi
- color#: line's color (1,15) (See "How to read the toolbox" for color codes)

**Examples:**
VGA_Hi 20 20 300 300 12 HORIZ_LINE

**See also:**
LS_LINE, HS_LINE, HORIZ_LINE, HORIZ_LINE_BIOS, VERT_LINE, VERT_LINE_BIOS, LINE_BIOS, ELLIPSE

**LINE_BIOS** 
(x1 y1 x2 y2 color# ...)

**Type:** WORD

**Category:** Video BIOS call drawing routine

**Purpose:**
Provide the capability of drawing a straight line in any orientation.

**Description:**
This routine draws a straight line in EGA/VGA graphics modes with the following inputs:

- x1: start point in the x-coordinate (0,640) if VGA_Hi
- y1: start point in the y-coordinate (0,480) if VGA_Hi
- x2: end point in the x-coordinate (0,640) if VGA_Hi
- y2: end point in the y-coordinate (0,480) if VGA_Hi
- color#: line's color (1,15) (See "How to read the toolbox" for color codes)

**Examples:**
VGA_Hi 20 20 300 300 12 HORIZ_LINE

**See also:**
HORIZ_LINE, HORIZ_LINE_BIOS, VERT_LINE, VERT_LINE_BIOS, LINE_BIOS, ELLIPSE
LS_LINE ( x1 y1 x2 y2 color# -- )

Type: CODE
Category: Direct-video drawing routine

Purpose:
Provide a straight line (slope <=1) drawing capability and supports the LINE routine.

Description:
This routine draws a straight line in EGA/VGA graphics modes with the following inputs:
x1 = start point in the x-axis (i.e., (0,640) if VGA_HI)
y1 = start point in the y-axis (i.e., (0,480) if VGA_HI)
x2 = end point in the x-axis (i.e., (0,640) if VGA_HI)
y2 = end point in the y-axis (i.e., (0,480) if VGA_HI)
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:
VGA_HI 100 20 300 12 HORIZ_LINE

See also:
VGA_HI, VERTICAL_LINE, VERT_LINE, LINE,

NORMAL_VIDEO ( -- )

Type: WORD
Category: Video environment controlling routine

Purpose:
Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:
NORMAL_VIDEO specifies the Data Rotate/Function Select register's (O5H) two bit fields to 00000000. This bit pattern forces latched pixels to be normalized when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:
VGA_HI OR_VIDEO 250 300 12 PUT_PIXEGA NORMAL_VIDEO

See also:
AND_VIDEO, XOR_VIDEO
OR_VIDEO

Type: WORD

Category: Video environment controlling routine

Purpose:

Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:

OR_VIDEO specifies the Data Rotate/Function Select register’s (DSW) two bit fields to 00000000. This bit pattern forces latched pixels to be OR’d when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:

VGA_HI OR_VIDEO 250 300 12 PUT_PIXEGA

See also:

AND_VIDEO, XOR_VIDEO, NORMAL_VIDEO

OUTTEXTXY

Type: WORD

Category: Direct-video graphic text drawing routine

Purpose: Write graphics text.

Description:

After VGA_HI is invoked, OUTTEXTXY will write texts in the quotes on the graphics screen with the following designation:

Input:

PTR # COUNT = put the texts in this form -> "$" abc..text =

color = color# (0,15) (See "How to read the toolbox" for color codes)

x = character position’s x-coordinate

y = character position’s y-coordinate

limx = bottom window clip = 0

limy = top window clip (i.e., 639 for horizontal orientation and 479 for vertical orientation)

hv = 0 vertical (top to bottom)

1 horizontal (left to right)

2 vertical (bottom to top)

3 horizontal (right to left)

Examples:

VGA_HI

$" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY
$" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY
$" HORIZONTAL 1" 13 300 200 0 369 1 OUTTEXTXY
$" HORIZONTAL 2" 14 300 200 0 369 3 OUTTEXTXY

See also:

TXT, TXT_BIOS, OUTTEXTXY_BIOS
OUTTEXTXY_BIOS  ( PTR # CRT COLOR XY LIMX LIMY FG -- )

Type: WORD
Category: Video BIOS call graphic text drawing routine
Purpose: Write graphics text.

Description:
After VGA.H1 is invoked, OUTTEXTXY_BIOS will write text inside the quotation marks (" " ) on the graphics screen with the following designation:

Input:
PTR # COUNT == put text in this form -> $" abc...text =

color == color# (0,15)  
(See "How to read the toolbox" for color codes)
  x == character position's x-coordinate  
  y == character position's y-coordinate  
      (0,639) for VGA.H1
l RX == bottom window clip (i.e. 639 for horizontal orientation and 479 for vertical orientation)
h v == 0 vertical (top to bottom)
      1 horizontal (left to right)
      2 vertical (bottom to top)
      3 horizontal (right to left)

Examples:
VGA_H1
$" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY_BIOS
$" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY_BIOS
$" HORIZONTAL 1" 13 300 200 0 349 1 OUTTEXTXY_BIOS
$" HORIZONTAL 2" 14 300 200 0 349 3 OUTTEXTXY_BIOS

See also:
TXT, TXT_BIOS, OUTTEXTXY

PUT_PIXEGA  ( x y color# -- )

Type: CODE
Category: Direct video pixel drawing assembly routine
Purpose:
Plot a point at the indicated pixel location

Description:
PUT_PIXEGA is a video BIOS call routine which plots a point with the color# (0,15) at the specified (x,y) coordinate pixel position.

Input:
  x == x-coordinate (0,639)
  y == y-coordinate (0,479)
  color# == color code numbers (0,15)
  (See "How to read the toolbox" for color codes)

Examples:
VGA_H1 40 50 12 PUT_PIXEGA

See also:
GET_PIXEGA
PUT_PIXEGA (equivalent BIOS call routine)
**PUT_PIXEL**

(x y color# -- )

*Type:* CODE

*Category:* Video BIOS function call 12

*Purpose:*
Plot a point at indicated pixel location

*Description:*
PUT_PIXEL is a video BIOS call routine which plots a point with the color# (0,15) at the specified (x,y) coordinate pixel position.

*Input:*
- x == x-coordinate (0,639)
- y == y-coordinate (0,479)
- color# == color code numbers (0,15)
  
  (See "How to read the toolbox" for color codes)

*Examples:*
- VGA_HI 40 50 12 PUT_PIXEL

*See also:*
- GET_PIXEGA
- PUT_PIXEGA (equivalent direct-video routine)

**RANDOMVECT**

( maxk k -- r1 .. rk )

*Type:* WORD

*Category:* Random number generator routine

*Purpose:*
Generate uniform density random numbers

*Description:*
Given the maximum value and the count, RANDOMVECT will generate integer random numbers in the ranges of (0,maximum value).

*Input:*
- maxk == maximum value in the set of random numbers
- k == count of random numbers

*Output:*
- r1 .. rk == random numbers

*Examples:*
- 50 5 RANDOMVECT will result in 30 45 34 2

*See also:*
READ_CHAR : ( -- char )

Type: CODE

Category: Video BIOS function call 8

Purpose:
Read character at current cursor position

Description:
READ_CHAR is a video BIOS call routine which
retrieves the character at the current cursor position.

Input: none
Output:
character (a, b, ... z, A, B, ...)

Examples:

See also:

---

SCROLL_PAGEDOWN: ( #oflines -- )

Type: CODE

Category: Video BIOS function call 7

Purpose:
Scroll the text on the screen

Description:
This function scrolls the text on the screen - lines move from
the top of the screen toward the bottom, and blank lines are
inserted at the top.

Input:
# of lines to scroll

Output:
scrolled text lines

Examples:
10 SCROLL_PAGEDOWN

See also:

SCROLL_PAGEUP
SCROLL_PAGEUP  ( #oflines -- )

Type: CODE

Category: Video BIOS function call 6

Purpose:
Scroll the text on the screen

Description:
This function scrolls the text on the screen - lines move from the bottom of the screen toward the top, and blank lines are inserted at the bottom.

Input:
# of lines to scroll

Output:
scrolled text lines

Examples:
10 SCROLL_PAGEUP

See also:
SCROLL_PAGEDOWN

SET_ACTIVEPAGE  ( page# -- )

Type: CODE

Category: Video BIOS function call 5

Purpose:
Selects Page as the active page for graphics output

Description:
This function sets Page as the active page for graphics output. Although the adapter may have several pages (or screens) of information in memory, only one page, the active display page, is visible at any one time. Most of the functions which allow screen modification (write characters, plot points, move the cursor, etc.) also allow selecting which page to modify, thus an invisible screen (non-active display page) may be changed. For example, while displaying one page, another may be created or changed. This feature allows switching to the new screen immediately (a technique useful for animation or slide shows). This function allows choosing which screen is displayed. Usually, screen 0 is the only screen displayed and modified.

Input:
page#

Output:

Examples:
3 SET_ACTIVEPAGE

See also:
SET_BORDER (color# --)

Type: CODE

Category: Video BIOS function call 16

Purpose:
Draw a border line with the new color selected

Description:
This function draws a border line with the color selected.

Input:
color# (0,15)
(See "How to read the toolbox" for color codes)

Output:

Examples:
12 SET_BORDER

See also:
SET_PALETTE

SET_GRAPHMODE (graphmode# --)

Type: CODE

Category: Video BIOS function call 0

Purpose:
Set the screen to various graphics mode

Description:
Sets the system to the specified graphics mode as shown below:

0 & 1 - 40 column alphanumeric (CGA compatible)
2 & 3 - 80 column alphanumeric (CGA compatible)
4 & 5 - 320x200 4-color graphics limited to 2 palettes (CGAHi)
6 - 640x200 2-color graphics; one must be black (CGAHi)
7 - monochrome alphanumeric (monochrome adapter compatible)
8 - 12 - reserved
13 - 320x200 16 color
14 - 640x200 16 color (EGAHi)
15 - 640x350 monochrome graphics (EGAMonoHi)
16 - 640x350 16 color (EGAHi)
17 - 640x480 monochrome graphics (VGAMonoHi)
18 - 640x480 16 color (VGAHi)
19 - 320x200 256 color (VGA only)

Input:
color# (0,15)
(See "How to read the toolbox" for color codes)

Output:
In effect, invoking this code clears the screen.

Examples:
18 SET_GRAPHMODE

See also:
8x16FONT
SET_PALETTE ( palette# color# ... )

Type: CODE

Category: Video BIOS function call 16

Purpose:
Change the color number entry setting in the palette to new color

Description:
This function updates a specified palette register to new color.

Input:

palette# == (0,15)
color# == (0,15)

(See "How to read the toolbox" for color codes)

Output:
...

Examples:
12 13 SET_PALETTE

See also:
SET_BORDER

SET_VPLANE ( Hex# ... )

Type: CODE

Category: Routine

Purpose:
Enable/disable video plane

Description:
SET_VPLANE allows only certain planes within the graphics memory to be drawn or changed. Works only when video mode is set to 0.

Possible Input:
0 == plane # 0
1 == plane # 1
2 == plane # 2
3 == plane # 3

Examples:
3 SET_VPLANE
**SIGPLOTB** (ptr # cnt color x y lim fg ...) 

**Type:** WORD  
**Category:** Direct-video graphic text drawing routine  
**Purpose:** Write graphics text.  
**Description:**  
After VGA_HI is invoked, SIGPLOTB will write text inside quotation marks (" ") on the graphics screen with the following designation:  
**Input:**  
PTR # COUNT = put the text in this form -> $"abc..text "  

color = color#(0,15)  
(See "How to read the toolbox" for color codes)  
x = character position's x-coordinate  
(0,79) for VGA_HI  
y = y-coordinate pixel position  
(0,479) for VGA_HI  
lim = top window clip" (i.e., 0 )  
FG = 0 = 00 = vertical (top to bottom), or  
1 = 01 = horizontal (left to right), or  
2 = 10 = vertical (top to bottom), store  
3 = 11 = horizontal (left to right), store  

**Examples:**  
VGA_HI  
$" VERTICAL 1= 11 40 200 0 0 $ SIGPLOTB  

**See also:**  
QUITEXT
TLINE (x y ..)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:
Demonstration of graphics drawing capability.

Description:
After VGA_HI is invoked, TLINE will draw random sets of straight lines in different colors cornered by (x,y).
This is a demo for the line drawing routine.

Examples:
VGA_HI 100 100 TLINE

See also:
FIREWORK2, FIREWORK3, TELLIPSE, TLINE_BIOS

TLINE_BIOS (x y ..)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:
Demonstration of graphics drawing capability.

Description:
After VGA_HI is invoked, TLINE_BIOS will draw random sets of straight lines in different colors cornered by (x,y).
This is a demo for the line drawing routine.

Examples:
VGA_HI 100 100 TLINE_BIOS

See also:
FIREWORK2, FIREWORK3, TELLIPSE, TLINE
Type: WORD

Category: Direct-video graphic drawing routine

Purpose:
   Demonstration of graphics text capability.

Description:
   After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.),
   is invoked and given (x,y) coordinates, TXT
   writes text "xxxx... " in four
   90-deg orientations centered at (300, 200).

Examples:
   VGA_HI TXT

See also:
   AMERICA, AMERICA_BIOS, TXT_BIOS

Type: WORD

Category: Video BIOS call graphic drawing routine

Purpose:
   Demonstration of graphics text capability.

Description:
   After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.),
   is invoked and given (x,y) coordinates, TXT_BIOS
   writes text "xxxx... " in four
   90-deg orientations centered at (300, 200).

Examples:
   VGA_HI TXT_BIOS

See also:
   AMERICA, AMERICA_BIOS, TXT
VERT_LINE (x y1 y2 color ...

Type: CODE

Category: direct-video drawing routine

Purpose:
Provide a vertical (slope=infinite) straight line drawing capability

Description:
This routine draws a vertical line in EGA/VGA graphics modes with the following inputs:
- x = horizontal position (0, 0, 639) if VGA_H1
- y1 = start point in the y-axis (0, 479) if VGA_H1
- y2 = end point in the y-axis (0, 479) if VGA_H1
- color# = line's color (1, 15)

(See "How to read the toolbox" for color codes)

Examples:
VGA_H1 100 20 300 12 VERT_LINE

See also:
HORIZ_LINE_BIOS, HORIZ_LINE, VERT_LINE_BIOS, LINE

VERT_LINE_BIOS (x y1 y2 color ...

Type: WORD

Category: Video BIOS call drawing routine

Purpose:
Provide a vertical (slope=infinite) straight line drawing capability

Description:
This routine draws a vertical line in EGA/VGA graphics modes with the following inputs:
- x = horizontal position (0, 0, 639) if VGA_H1
- y1 = start point in the y-axis (0, 479) if VGA_H1
- y2 = end point in the y-axis (0, 479) if VGA_H1
- color# = line's color (1, 15)

(See "How to read the toolbox" for color codes)

Examples:
VGA_H1 100 20 300 12 VERT_LINE_BIOS

See also:
HORIZ_LINE_BIOS, HORIZ_LINE, VERT_LINE_BIOS, LINE
**VGA_HI (..)**

**Type:** WORD

**Category:** Video screen environment setup routine

**Purpose:**
Set the system to the specified graphics mode

**Description:**
Invoking VGA_HI will bring the screen to high-resolution-EGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

- 640x480 16-color graphics
- It also sets up the graphics character heights to be 16 units (pixels) high. (EGA_HI has 14 units high character size)

During the process, the screen gets cleared.

**Examples:**

```
VGA_HI 50 50 200 200 12 LINE_BIOS
```

**See also:**

- EGA_HI, CGA_LO, EGA_LO, EGA_HI, VGA_LO

---

**WRITE_CHAR (char ..)**

**Type:** CODE

**Category:** Video BIOS function call 9 and 10

**Purpose:**
Write character at current cursor position

**Description:**
WRITE_CHAR is a video BIOS call routine that writes a string of characters at current cursor position.

**Input:**
- character

**Examples:**

```
abc WRITE_CHAR
```

**See also:**

- WRITE_TCHAR
WRITE_TCHAR ( char -- )

Type: CODE

Category: Video BIOS function call 14

Purpose:
Write character at current cursor position and move the cursor to next position.

Description:
WRITE_TCHAR is a video BIOS call routine which writes character at current cursor position and the cursor is moved to the next position. Unlike the other write character functions, this function interprets the bell, carriage return, and linefeed characters as commands rather than characters from the IBM set.

Input:
character

Examples:
abc WRITE_TCHAR

See also:
WRITE_CHAR

XOR_VIDEO ( -- )

Type: WORD

Category: video environment controlling routine

Purpose:
Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:
XOR_VIDEO specifies the Data Rotate/Function Select register's (03H) two bit fields to 00011000. This bit pattern forces latched pixels to be XORred when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:
VGA_HI XOR_VIDEO 250 300 12 PUT_PIXEGA

See also:
AND_VIDEO, OR_VIDEO
APPENDIX B

FORTH GRAPHICS TOOLBOX
SOURCE CODE

.B-1/B-2
memory. Only one page is visible at any one time; this is called
active display page. Most of the functions which allow you to modify
the screen (write characters, plot points, move the cursor, etc.) also
let you choose which page to modify and thus an invisible screen may
be changed. Through this feature, you may display one page while another
is being created, and then immediately switch to the new screen (a
technique useful for animation or slide shows). This function lets you
choose which screen is displayed. Usually, screen 0 is the only screen
displayed and modified.

code getxy ( ... x y )
// function call 3 ( also known as rdcurpos )
// Get current (x,y) cursor position
MOV AX, # 200
INT # 10
NEXT END-Code

code getxy ( ... x y )
// function call 4: read light pen position
// VGA does not support a light pen

code setactivepage ( ... page# ... )
// function call 5
// Selects page as the active page for graphics output.
// The adapter may have several pages (or screens) of information in
This function call is used to plot a point on the screen.

\[ \text{POP AX, POP DX, POP CX} \]
\[ \text{XOR BH, BH OR AH, # OOOH} \]
\[ \text{INT # 010 \quad END-CODE} \]

\[ \text{CODE GET_PIXEL ( x, y -- color# )} \]
\[ \text{function call 13} \]
\[ \text{Reads the color# of the pixel at (x,y)} \]

\[ \text{POP DX, POP CX} \]
\[ \text{MOV AH, # OOOH XOR BH, BH} \]
\[ \text{INT # 010} \]
\[ \text{XOR AH, AH PUSH AX} \]
\[ \text{NEXT END-CODE} \]

\[ \text{CODE WRITE_TCHAR ( char -- )} \]
\[ \text{function call 14} \]
\[ \text{A character is written and the cursor is moved to the next position.} \]
\[ \text{Unlike the other write character functions, this function interprets} \]
\[ \text{the bell, carriage return, and linefeed characters as commands rather} \]
\[ \text{than characters from the IBM set.} \]

\[ \text{POP AX} \]
\[ \text{MOV AH, # OOOH XOR BX, BX} \]
\[ \text{INT # 010 \quad END-CODE} \]

\[ \text{CODE GET_GRAPHMODE ( -- video mode )} \]
\[ \text{function call 15} \]
\[ \text{Returns information about the current graphics mode setting.} \]

\[ \text{MOV AX, # 0FO0 INT # 010} \]
\[ \text{XOR AH, AN \quad ( VIDEO MODE # = AL )} \]
\[ \text{PUSH AX} \]
\[ \text{NEXT END-CODE} \]

\[ \text{CODE SET_PALETTE ( palette# color# -- )} \]
\[ \text{function 16} \]
\[ \text{Changes the setting of the color number entry in the palette} \]
\[ \text{to new color} \]

\[ \text{POP CX, POP BX} \]
\[ \text{MOV BX, CL MOV AX, # 01000} \]
\[ \text{INT # 010 \quad END-CODE} \]

\[ \text{CODE SET_BORDER ( color# -- )} \]
\[ \text{function call 16 \quad Subservice call 0th VGA/EGA graphics only} \]
\[ \text{Changes the setting of the color number entry in the palette} \]
\[ \text{to new color} \]

\[ \text{POP CX} \]
\[ \text{MOV BH, CL MOV AX, # 01001} \]
\[ \text{INT # 010 \quad END-CODE} \]

\[ \text{CODE GEN_CHAR ( code# -- )} \]
CODE GET PIXEGA ( x y -- color )
POP AX
POP BX ( AX -> y, BX -> x )
CALL EGPAX_ADDR
MOV CX, AH
SHL CX, CL
PUSH SI
MOV SI, BX
XOR BL, BL
MOV DX, # O5CE
MOV AX, # 0504
BEGIN
( LOOP THRU ALL 4 VIDEO PLANES )
OUT DX, AX
MOV BH, ES: [SI] ( BX -> BIT PLANE VALUE )
AND BH, CH ( MASK ONE BIT )
NEG BX ( BIT = 1 IF MASK = 1 ELSE = 0 )
ROL BX, # 1 ( BL -> NEXT BIT PLANE VALUE )
DEC AN ( NEXT BIT PLANE VALUE )
< UNTIL
POP SI
MOV AL, BL
XOR AH, AH
PUSH AX ( AL -> COLOR )
CALL EGA_RESET
NEXT END-CODE

CODE SET VPLANE ( Hex# -- )
( WORKS IN VIDEO MODE = 0 ONLY )
POP AX
MOV AH, AL
MOV AL, # 0 ( BIT 0 - PLANE 0 )
MOV AL, # 1 ( BIT 1 - PLANE 1 )
MOV DX, # O5CA
MOV AL, # 0 ( 2 - PLANE 2 )
OUT DX, AX
MOV AL, # 3 ( 3 - PLANE 3 )
NEXT END-CODE
( 1 = ENABLE, 0 = DISABLE )

B-5

****** DRAW VERTICAL LINE **********

CODE VERT LINE ( x y1 y2 color -- )
POP BX
GET COLOR
MOV DX, # O5CE ( SET UP GRAPHICS CONTROLLER )
MOV AH, BL ( USE WRITE MODE 0 )
XOR AH, AL ( ENABLE ONLY PLANES AFFECTED )
OUT DX, AX
MOV AX, # 0001 ( ENABLE PLANES )
OUT DX, AX
MOV AH, PIXEL_MODE ( PIXEL WRITE MODE )
MOV AL, # 3 ( 3 )
OUT DX, AX
POP BX POP AX \ y2, y1
MOV CX, BX
SUB CX, AX \ y = Y2-Y1
< IF
NEG CX
MOV AX, BX
THEN
INC CX
POP BX \ GET X LOCATION
PUSH CX \ SAVE COUNTER = y -> Y2-Y1
CALL EGPAX_ADDR
SHL AH, CL \ MASK BITS IN BYTE
MOV AL, # 8
OUT DX, AX
POP CX DEC CX
PUSH SI
MOV SI, # 050 \ HEX 50 = 80 BYTES/LINE
BEGIN
DRAW LINE LOOP OR ES: [BX], AL \ CX IS y COUNTER

****** DRAW HORIZONTAL LINE **********

CODE HORIZ LINE ( y x1 x2 color -- )
POP AX
GET COLOR
MOV DX, # 03CE ( SET UP GRAPHICS CONTROLLER )
MOV AH, BL ( USE WRITE MODE 0 )
XOR AH, AL ( ENABLE ONLY PLANES AFFECTED )
OUT DX, AX
MOV AX, # OF01 ( ENABLE PLANES )
OUT DX, AX
MOV AH, PIXEL_MODE ( PIXEL WRITE MODE )
MOV AL, # 3 ( 3 )
OUT DX, AX
POP CR POP BX POP AX \ x2, x1, y
MOV 0 [BP], BX \ 0 [BP] = x1
MOV 2 [BP], CX \ 2 [BP] = x2
SUB CX, BX \ FORCE x1 < x2
< IF
MOV BX, 2 (BP)
XCHG BX, 0 [BP]
MOV 2 (BP), BX
MOV BX, 0 [BP]
THEN
CALL EGPAX_ADDR
MOV DL, BX \ ES:DI -> BUFFER
MOV DH, AH \ DN = UNSHIFTED BIT MASK FOR LEFTMOST BYTE
NOT DX
SHL DX, CL \ DN = REVERSE BIT MASK FOR FIRST BYTE
MOV DX, 2 [BP] \ DN = BIT MASK FOR FIRST BYTE
MOV CX, 2 [BP] \ CX = x2
AND CL, # 7 \ CL = NUMBER OF BITS TO SHIFT LEFT
MOV DL, # OFF \ DL = UNSHIFTED BIT MASK FOR RIGHTMOST BYTE
SHL DL, CL \ DL = BIT MASK FOR LAST BYTE
MOV AX, 2 [BP] \ AX = x2
MOV BX, 0 [BP] \ BX = x1
MOV CX, # 0 \ NUMBER OF BITS TO SHIFT TO CONVERT PIXELS
SHR AH, CL \ BYTE OFFSET OF X2
SHR BX, CL \ BYTE OFFSET OF X1
MOV CX, AX \ BX = X1
MOV BX, DX \ BX = X1
MOV DX, # 03CE \ DX = GRAPHICS CONTROLLER PORT
MOV AL, # 8 \ AL = BIT MASK REGISTER
PUSH ES
POP DS
MOV SI, DI \ VIDEO BUFFER -> DS:SI
NAVSWC MP 91-404

06/13/1991 12:45   Filename: GRAPHICS.SEO   Page 11

06/13/1991 12:45   Filename: GRAPHICS.SEO   Page 12

{ AX = u2 [lo-order word] )
  { CX = vl [16-bit number ] )
  ( returns: DX:AX = 32-bit result )

PUSH AX
  ( preserve u2 )
MOV AX, DX
  ( AX=v1 )
MUL CX
  ( AX=hi-order word of result )
XCHG AX, CX
  ( AX=v1, CX=hi-order word )
PUSH DX
  ( DX=u2 )
MUL DX
  ( AX=lo-order word of result )
POP DX
  ( DX=carry )
ADD DX, CX
  ( CX=hi-order word of result )
RET

LABEL SET4PIXELS
  ( this subroutine used in CODE ELLIPSE )
  ( call with CH= y- increment [0,-1] )
  ( CL= x- increment [0,1] )

PUSH AX
  ( preserve these regs )
PUSH BX
PUSH DX

HEX

MOV DX, # 03CE
  ( DX= graphics controller port )
XOR BX, RX
  ( BX = 0 )
TEST CH, CH
  ( jump if y-increment =0 )

DECIMAL

MOV BX, # 0080
  ( BX= positive increment )
NEG BX
  ( BX= negative increment )
THEN

MOV AL, # 8
  ( AL= Bit Mask reg number )

\ pixels at (xc-x, yc+y) and (xa-x, ya-y)

\ pixels at (xc-x, yc+y) and (xa-x, ya-y)

\ pixels at (xs-x, ys+y) and (xa-x, ya-y)

\ pixels at (xs-x, ys+y) and (xa-x, ya-y)

\ pixels at (xs-x, ys+y) and (xa-x, ya-y)
MOV AX, # 0000 (AN = value for enable set/reset)
(CALL bit planes enabled)
OUT DX, AX (AL = enable set/reset reg number)

\ Initial constants

DECIMAL
MOV AX, 6 (BP)
MUL AX
MOV -28 (BP), AX
MOV -30 (BP), DX (a'2)
RCL DX, # 0001
MOV 56 (BP), AX
MOV -38 (BP), DX (2*a'2)

MOV AX, 8 (BP)
MUL AX
MOV -32 (BP), AX
MOV -34 (BP), DX (b'2)
RCL DX, # 0001
MOV -40 (BP), AX
MOV -42 (BP), DX (2*b'2)

\ plot pixels from (0,0) until dx/dy = 1
\ initial buffer address and bit mask

MOV AX, # 0B0 (AL = video buffer line length)
MUL B (BP) (AX = relative byte offset of b)
MOV S1, AX
MOV D1, AX

MOV AX, 4 (BP) (AX = yc)
MOV DX, 2 (BP) (BX = xc)
CALL EGAPIX_ADDR (AN = bit mask)
CALL EIPIX_ADDR (ES:BX = buffer)
(CL = Wbits to shift left)

MOV AH, # 1 (AH = bit mask for first pixel)
SRL AH, CL
MOV -10 (BP), AH
MOV -12 (BP), AH
ADD S1, BX
MOV -2 (BP), SI
MOV -4 (BP), SI
SUB BX, D1
MOV -6 (BP), BX
MOV -8 (BP), BX

\ Initial decision variables

XOR AX, AX
MOV -20 (BP), AX
MOV -22 (BP), AX (dx = 0)
MOV AX, -36 (BP)
MOV DX, -38 (BP)
MOV CX, 8 (BP)
CALL LONGMULTIPLY (perform 32-bit by 16-bit multiply)
MOV -24 (BP), AX
MOV -26 (BP), DX (dy = 2*a'2 * b)

\ loop until dy/dx = -1

MOV BX, B (BP) (BX = initial y-coordinate)
XOR CX, CX (CH = 0 [initial y-increment])
(CL = 0 [initial x-increment])

BEGIN
MOV AX, -20 (BP)
MOV DX, -22 (BP)
SUB AX, -24 (BP)
SUB DX, -26 (BP)

0< WHILE (jump if dx=dy)
CALL SETPIXELS
MOV CX, # 0001 (CH = 0 [y-increment])
(CL = 1 [x-increment])
CMP -18 (BP), # 0000

0> IF (jump if d < 0)
MOV CH, # 1 (increment in y direction)
DEC BX (decrement current y-coordinate)
MOV AX, -24 (BP)
MOV DX, -26 (BP)
MOV -24 (BP), DX (dy = dy - 2*a'2)
MOV -26 (BP), DX (dy = 2*a'2)
SUB -16 (BP), AX
SUB -18 (BP), DX (d = d - dy)
ELSE
MOV AX, -20 (BP)
MOV DX, -22 (BP)
ADD AX, -32 (BP)
ADD DX, -34 (BP)
MOV -20 (BP), AX
MOV -22 (BP), DX (dx = dx + 2*b'2)
ADD AX, -32 (BP)
ADD DX, -34 (BP) (DX:AX = dx + b'2)
ADD -16 [BP], AX
ADC -18 [BP], DX  ( d += dx + b·2 )
THEN
REPEAT
\ plot pixels from current (x,y) until y < 0
\ initial buffer address and bit mask
PUSH BX              ( preserve current y-coordinate )
PUSH CX              ( preserve x- and y-increments )
MOV AX, -28 [BP]
MOV DX, -30 [BP]
SUB AX, -32 [BP]
SBB DX, -34 [BP]    ( DX:AX = a·2 - b·2 )
MOV BX, AX
MOV CX, DX
SAR DX, # 0001      ( DX:AX = (a·2 - b·2) / 2 )
RCR AX, # 0001      ( DX:AX = 3*(a·2 - b·2)/2 )
ADD AX, BK
ADD DX, CX
SUB AX, -20 [BP]
SBB DX, -22 [BP]
SUB AX, -24 [BP]
SBB DX, -26 [BP]    ( DX:AX = 3*(a·2 - b·2)/2 - (dx+dy) )
SAR DX, # 0001      ( DX:AX = (3*(a·2 - b·2)/2 - (dx+dy))/2 )
RCR AX, # 0001
ADD -16 [BP], AX
ADD -18 [BP], DX    ( update d )
\ loop until y < 0
POP CX              ( CX,CL = y- annd x-increments )
POP BX              ( BX xy )

BEGIN
CALL SET4PIXELS
MOV CX, # 0100      ( CH = 1 y-increment )
( CL = 0 x-increment )

DECIMAL
CMP -18 [BP], # 0000
0< IF                  ( jump if d >= 0 )
MOV CL, # 1
( increment in x direction )
MOV AX, -20 [BP]
MOV DX, -22 [BP]
ADD AX, -40 [BP]
ADD DX, -42 [BP]    ( DX:AX = dx + 2*b·2 )
MOV -20 [BP], AX
MOV -22 [BP], DX    ( dx += 2*b·2 )
ADD -16 [BP], AX
ADD -18 [BP], DX    ( d += dx )
THEN
ELSE GET_PIXEL
THEN ;

: 8x14FONT ( ... )
O200 FONTAD FONTZ 21 0E CHRMGT 1 ;

: Bx16FONT
O200 FONTAD FONTZ 21 010 CHRMGT 1 ;

: Bx8FONT
O200 FAGE FONTZ 21 08 CHRMGT 1 ;

: WPLANE
2 364 PCI 365 PCI ;

DECIMAL

: EGA_HI 16 SETGRAPHMODE 8x14FONT ;
: EGA_LO 14 SETGRAPHMODE 8x14FONT ;
: VGA_HI 18 SETGRAPHMODE 8x16FONT ;
: VGA_LO 4 SETGRAPHMODE ;
: CGA_HI 6 SETGRAPHMODE ;
: CGA_LO 2 SETGRAPHMODE ;

: COLOR ON

: CHRPLOT
F ( CHRN COLOR X Y LIML LINK NV / TBADR # DX PFLG )
\ K X is 0 to 659, Y is 0 to 349, COLOR \ I LIML and LIMH are top and bot of window clip limits \ DIR : 0.2 -VERTICAL 1.3 -HORIZONTAL

NV 1 AND
IF X Y TO X Y THEN ;

NV DUP 0 = SNAP 3 = OR
IF 1 TO PFLG ELSE 0 TO PFLG
THEN ;

FONZ PFLG CHRMGT * + TO TBADR

CHRMGT 0
DO X Y PFLG
IF NEGATE
THEN + DUP DUP TO DX
LIML < SNAP LIMH > OR
TBADR 1 + CBL DUP D = ROT OR
IF DROP ELSE 128 8 0
DO 2DUP AND
IF DX Y 1
CASE NV 0 = IF +
ELSE NV 2 = IF -
ELSE NV 3 = IF NEGATE + SNAP
ELSE + SNAP
ENDCASE
THEN color PUT_PIXEL
THEN LOOP ;

: OUTTEXTXY
F ( PTR # CNT COLOR X Y LIML LINK FG )
\ input - string pointer and length
CMT 0
DO PTR 1 + CBL
COLOR X 1 8
CASE FG 3 = IF NEGATE + Y
ELSE FG 0 = IF Y SNAP +
ELSE FG 2 = IF Y SNAP -
ELSE + Y
ENDCASE
THEN LIML LINK FG CHRPLOT

: OUTTEXTXY BIOS
F ( PTR # CNT COLOR X Y LIML LINK FG )
\ input - string pointer and length
CMT 0
DO PTR 1 + CBL
COLOR X 1 8
CASE FG 3 = IF NEGATE + Y
ELSE FG 0 = IF Y SNAP +
ELSE FG 2 = IF Y SNAP -
ELSE + Y
ENDCASE
THEN LIML LINK FG CHRPLOT BIOS

: SGPLOTB
F ( PTR # CNT COLOR X Y LIML LINK FG / TBADR # XC YC )
\ X is 0 to 79, Y is 0 to 349
\ FG: 0 bit is vert/horis; 1 bit is STORE or OR
FG 2 AND IF \ HEX ) 3 36C PCI 10 36F PCI ( DECIMAL )
THEN CMT 0
DO PTR 1 + CBL
FONZ ROT CHRMGT * + TO TBADR
COLOR WPLANE
08/13/1991 12:45  Filename: GRAPHICS.SEO  Page 19

FG 1 AND
IF  X 1 * Y
ELSE  X Y 1 CMWGT * +
THEN 466 MIN TO YC 0 MAX 79 MIN TO XC
14 O
DO TBADR 1 + CAL
YC 1 + DUP LIN DUP O<
IF NEGATE >
ELSE <
THEN 0=
IF IBM B0 + XC + 2DUP CAL DROP CIL
THEN
LOOP
LOOP ( NEX ) 3 3CE PC! 0 3CF PC! [ DECIMAL ] ;
: LINE FC ( X1 Y1 X2 Y2 COLOR )
CASE X1 X2 = \ VERTICAL LINE x1 = x2
IF X1 Y1 Y2 COLOR VERT Line -
ELSE Y1 Y2 = \ TEST FOR HORIZ LINE y1 = y2
IF Y1 X1 X2 COLOR HORIZ_LINE \ dy = dx
ELSE Y1 Y2 - ABS X1 X2 - ABS
> IF X1 X1 Y2 X2 COLOR HS_LINE \ dx = dy - SLOPE > 1
ELSE X1 X1 Y2 X2 COLOR LS_LINEAR \ dx = dy - SLOPE < 1
THEN
ENDCASE ;
TOMODE ON
: LINE_BIOS ( x1 y1 x2 y2 color# / m1 m2 )
x2 x1 + to m1 y2 y1 - to m2
m1 0= if m2 0< if y2 y1 do i x1 swap color# put_pixel -1
+loop
else m2 0< if x1 x1 y1 y1 color# put_pixel
else y2 y1 do i x1 swap color# put_pixel
loop
then
then
else x2 x1 > if m1 0 do i DUP x1 + swap m2 m1 */ y1 +
COLOR# put_pixel
loop
else m2 0< if m1 0 do i DUP x1 + swap m2 m1 */ y1 +
COLOR# put_pixel -1
+loop
else m1 0 do i DUP x1 + swap m2 m1 */ abs
y1 +
COLOR# put_pixel -1
+loop
then
THEN ;
: VERT_LINE_BIOS ( x1 y1 y2 color# ... )
F ( x x1 y2 color# )
X X1 X2 COLOROP LINE_BIOS ;
: HORIZ_LINE_BIOS ( y1 x1 x2 color# ... )
F ( y y1 x2 color# )
X1 X2 Y COLOR OP LINE_BIOS ;
AXISPARAM HAXIS VAXIS
2 DUP TO HAXIS S_LEN TO VAXIS S_LEN
4 DUP TO HAXIS L_LEN TO VAXIS L_LEN
7 DUP TO HAXIS A_COLOR TO VAXIS A_COLOR
7 DUP TO HAXIS A_COLOR
5 3 TO HAXIS M_SPACE TO HAXIS N_SPACE
TYPE> SCALPARAM
INTEGER SM_TIC LG_TIC NUMERIC PTS
DOUBLE_REAL _G_OF _L_OFS _N_OFS
ENTTYPE> SCALPARAM
SCALPARAM SCALER1
TYPE> GRIDPARAM
INTEGER G_COLOR G_LEN
ENTTYPE> GRIDPARAM
GRIDPARAM HGRID VGRID
100 DUP TO HGRID G_LEN TO VGRID G_LEN
7 DUP TO HGRID A_COLOR TO VGRID A_COLOR
VARIABLE SHOWGRID
SHOWGRID OFF
: INT ( dr -- dr+ )
NIP 0 SWAP ;
: TRUNC ( dr -- n )
NIP ;
: ROUND ( dr -- n )
SWAP 0 < ;
: FIXPERR
" FIXED POINT ARITHMETIC ERROR IN " R>
BEGIN 2- DUP >NAME 1 LITERAL <>
UNTIL >NAME DUP YG? ?LINE .10 QUIT ;
: 10^N ( n -- 10^n )
R> 10 1 R8 ABS DUP 5 <
IF 0 ?DO OVER *
LOOP NIP R> 0<
ELSE 0 SWAP
THEN
ELSE 0 SWAP
THEN
: *10^N ( dr n -- dr+ )
10^N DR+ ;
: /10^N ( dr n -- dr )
NEGATE *10^N ;
: CALTIC OFFS \ INPUT ( X1,SM_OFS TIC_locs, TIC )
F ( ST VAL # SM_OFS # TIC LOC TICchia ) -
ST VAC SM_OFS 0+ TRUNC TO TEMP
0 TO CT
BEGIN TEMP TIC_LOC MOD 0=
WHILE TEMP TIC Loc + TO TEMP
REPEAT
SM_OFS 0 CT TIC Loc 0+ ;
: CALPOWER
F(DX # THOLD # / EXP ) \ INPUT = dx * OUTPUT = exponent
0 TO EXP
DX THOLD
D< IF BEGIN EXP DECR
DX THOLD EXP *10^W D > EXP ^4 < OR
UNTIL ELSE BEGIN EXP INCR
DX THOLD EXP *10^W D > EXP ^3 < OR
UNTIL EXP DECR
THEN
EXP ^4 < EXP ^3 < OR
IF CORD CR FASSTYPE EXP :
TRUE THEN \ ABORT^ ERR1 - AUTOSCALER OUT OF RANGE " \ AB2
THEN
EXP ;

AUTO_RANGE
F( DX # THOLD # / EXP ) \ INPUT : dx, 10^W OF dx
DX THOLD CALPOWER DUP TO EXP \ OUTPUT: SM_TIC, LG_TIC, NUMERIC
CASE DX 2.0 THOLD DR* EXP *10^W D*:
1 IF \ RANGE 1 : dx < 60.0 * 10^W
7 \ SM_TIC INTERVAL
5 \ LG_TIC INTERVAL
10 \ NUMERIC INTERVAL
ELSE DX 4.0 THOLD DR* EXP *10^W D*:
1 IF \ RANGE 2 : dx < 120.0 * 10^W
0 20 20
ELSE 5 10 50 \ RANGE 3 : dx < 300.0 * 10^W
ENDCASE ;

START_LOC
"F ( P # OFS # LOC )
P OFS DR* 0 LOC 0 ;

INCR_TIC
F ( I TIC N DXP # T START # )
0 I TIC * DXP DR/ 0 W DR* T START D + ROUND ;

TIC_PARAM
F( DX # EXP LOC L1 # P ' SCALERPARAM / DXP # )
DX EXP /10^W 200UP \ DXP
0 P PTS 2SWAP DR/ TO DXP
DXP P SWAP LOC START LOC
\ L START
DXP P N OFS LOC START LOC \ N START
DXP P N OFS LOC START LOC \ L1 EXP 7/10^W P N OFS D* EXP *10^W ; \ N.VAL

DRAWGRID
F( G START # LG_TIC DXP # N DN L1 HV GP ' GRIDPARAM
/ LASTPRT LEN KOLOR )
N DN TO LASTPT
GP C_LEN TO LEN
GP G_COLOR TO COLOR
DXP 0 LG_TIC DR/ TRUNC 1+ 0 \ NUMBER OF GRID LINES TO PLOT
DO T LG_TIC ON DXP G START INCR_TIC DUP DUP
LASTPT < SWAP N T AND IF
IF L1 DUP LEN HV
IF - KOLOR VERT_LINE / HV - GRID ORIENTATION
ELSE + KOLOR HORIZ_LINE \ T : VERTICAL
THEN \ 0 : HORIZONTAL
ELSE DROP
THEN
LOOP ;

DRAWGRID_BIOS

PLOTTIC
F( DXP # S START # L START # SM_TIC LG_TIC PTS LASTPT X Y HV
AP ' AXISPARAM / T_LOC L_LOC K KOLOR SL LL )
AP A_COLOR TO KOLOR
AP S_LEN TO SL
AP L_LEN TO LL
IF Y X LASTPT KOLOR HORIZ_LINE
X Y TO X TO Y ( SWAP X AND Y FOR VERTICAL TIC MARKS )
ELSE X Y LASTPT KOLOR VERT_LINE
THEN
THEN
X Y TO X TO Y
\ STAGE 2
L_START ROUND TO L_LOC
DXP 0 SM_TIC DR/ TRUNC 1+ 0
DO T SM_TIC PTS DXP S/start INCR_TIC DUP TO T_LOC
L_LOC =
IF T_LOC DUP Y > SWAP LASTPT AND IF
IF T_LOC X DUP HV
IF LL + KOLOR VERT_LINE / PLOT LARGE TIC
ELSE LL + KOLOR HORIZ_LINE
THEN
THEN
K LG_TIC PTS DXP L_START INCR_TIC TO L_LOC
K INCR
ELSE T LOC DUP Y > SWAP LASTPT AND IF
IF T LOC X DUP HV
IF SL - KOLOR VERT_LINE / PLOT SMALL TIC
ELSE SL + KOLOR HORIZ_LINE
THEN
THEN
THEN

PLOTTIC_BIOS
F( DXP # S START # L START # SM_TIC LG_TIC PTS LASTPT X Y HV
AP ' AXISPARAM / T_LOC L_LOC K KOLOR SL LL )
AP A_COLOR TO KOLOR
AP S_LEN TO SL
AP L_LEN TO LL
IF Y X LASTPT KOLOR HORIZ_LINE BISO
X Y TO X TO Y ( SWAP X AND Y FOR VERTICAL TIC MARKS )
ELSE X Y LASTPT KOLOR VERT_LINE_BIOS
THEN
THEN
X Y TO X TO Y
\ STAGE 2
L_START ROUND TO L_LOC
DXP 0 SM_TIC DR/ TRUNC 1+ 0
DO T SM_TIC PTS DXP S/start INCR_TIC DUP TO T_LOC
L_LOC =
IF T_LOC DUP Y > SWAP LASTPT AND IF
IF T_LOC X DUP HV
\textbf{SHOWGRID}
\begin{verbatim}
IF L_START P LG_TIC EXP Y P PTS X 0 VGRID THEN :

: AUTOSCALE
F( X1 # X2 # X Y COLOR NPT HV / DX # EXP S_VAL # )
X1 X2 X1 D - 20UP TO DX
60.0 NPT - 60 */ THRESHOLD VALUE
AUTO RANGE TO SCALER1 NUMERIC \ AUTORANGE INPUT - DX, THRESHOLD
TO SCALER1 LG_TIC
TO SCALER1 SM_TIC
TO EXP
X1 EXP /10^N TO S_VAL \ S_VAL = X1 / P NTH
S_VAL 0 SCALER1 SM_TIC/OR INT 1.0 D+ 0 SCALER1 SM_TIC OR* S_VAL D-
TO SCALER1 S_OFS
S_VAL SCALER1 S_OFS SCALER1 LG_TIC SCALER1 SM_TIC CALTIC_OFS
TO SCALER1 L_OFS
S_VAL SCALER1 S_OFS SCALER1 NUMERIC SCALER1 SM_TIC CALTIC_OFS
TO SCALER1 W_OFS

\textbf{B-1-5}
\end{verbatim}
\textbf{AUTOSCALE BIOS}
\begin{verbatim}
F( X1 # X2 # X Y COLOR NPT HV / DX # EXP S_VAL # )
X2 X1 X1 D - 20UP TO DX
60.0 NPT - 60 */ THRESHOLD VALUE
AUTO RANGE TO SCALER1 NUMERIC \ AUTORANGE INPUT - DX, THRESHOLD
TO SCALER1 LG_TIC
TO SCALER1 SM_TIC
TO EXP
X1 EXP /10^N TO S_VAL \ S_VAL = X1 / P NTH
S_VAL 0 SCALER1 SM_TIC/OR INT 1.0 D+ 0 SCALER1 SM_TIC OR* S_VAL D-
TO SCALER1 S_OFS
S_VAL SCALER1 S_OFS SCALER1 LG_TIC SCALER1 SM_TIC CALTIC_OFS
TO SCALER1 L_OFS
S_VAL SCALER1 S_OFS SCALER1 NUMERIC SCALER1 SM_TIC CALTIC_OFS
TO SCALER1 W_OFS

\textbf{TEST AND EXAMPLE WORDS}
\end{verbatim}
\textbf{AST}
\begin{verbatim}
F( X1 # X2 # ) VGA HI
SHOWGRID ON
80 TO NGRID G LEN
X1 X2 X1 0 100 12 15 256 1 AUTOSCALE
X1 X2 0 300 14 16 128 1 AUTOSCALE
80 TO NGRID G LEN
X1 X2 530 20 15 320 0 AUTOSCALE
X1 X2 400 160 11 160 0 AUTOSCALE

\textbf{VAR: RNSTATE 1}
\end{verbatim}
\textbf{URANS}
\begin{verbatim}
( AD N - - )
RNSTATE @ - ROT 0
DO \ rot
\textbf{UNIFORM}
\textbf{SNAP R = 1+}
\textbf{LOOP DROP R NSTATE 1 ;}

\textbf{RANDOMCONNECT ( max k r - r1 ... pk )}
RNSTATE @ SNAP 0
DO \ max
\textbf{UNIFORM R}
\textbf{OVER UN R SNAP R} \ \ max \ max \ max
\textbf{LOOP R NSTATE 1 DROP ;}

\textbf{SAMPLE DEMO GRAPHICS}
<table>
<thead>
<tr>
<th>DISTRIBUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Copies</strong></td>
</tr>
<tr>
<td>Chief of Naval Operations</td>
</tr>
<tr>
<td>Department of the Navy</td>
</tr>
<tr>
<td>ATTN: OP-392C</td>
</tr>
<tr>
<td>Washington, DC 20350</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>COMDESRON-15</td>
</tr>
<tr>
<td>Attn: LCD Steve Anthony</td>
</tr>
<tr>
<td>FPO San Francisco CA 96601-4717</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>COMDESRON-20</td>
</tr>
<tr>
<td>Attn: CAPT Kaiser</td>
</tr>
<tr>
<td>FPO Miami FL 34099-4719</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td>Cameron Station</td>
</tr>
<tr>
<td>Alexandria, VA 22304-61451</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>REPORT DOCUMENTATION PAGE</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td><strong>PUBLIC REPORTING BURDEN</strong></td>
</tr>
<tr>
<td><strong>FOR THE COLLECTION OF INFORMATION</strong></td>
</tr>
<tr>
<td><strong>IS ESTIMATED TO AVERAGE 1 HOUR PER RESPONSE</strong></td>
</tr>
<tr>
<td><strong>INCLUDING THE TIME FOR REVIEWING INSTRUCTIONS, SEARCHING EXISTING DATA SOURCES, GATHERING AND MAINTAINING THE DATA NEEDED, AND COMPLETING AND REVIEWING THE COLLECTION OF INFORMATION.</strong></td>
</tr>
<tr>
<td><strong>SEND COMMENTS REGARDING THE BURDEN ESTIMATE OR ANY OTHER ASPECT OF THE COLLECTION OF INFORMATION, INCLUDING SUGGESTIONS FOR REDUCING THE BURDEN, TO WASHINGTON HEADQUARTERS SERVICES, DIRECTORATE FOR INFORMATION OPERATIONS AND REPORTS, 1215 JEFFERSON DAVIS HIGHWAY, SUITE 1204, ARLINGTON, VA 22202-4302, AND TO THE OFFICE OF MANAGEMENT AND BUDGET, PAPERWORK REDUCTION PROJECT (0704-0188), WASHINGTON, DC 20503.</strong></td>
</tr>
</tbody>
</table>

**1. AGENCY USE ONLY (Leave blank)** | **2. REPORT DATE** | **3. REPORT TYPE AND DATES COVERED** | **5. FUNDING NUMBERS**
---|---|---|---
**4. TITLE AND SUBTITLE** | **14 June 1991** | **FINAL** | **NA**
**FORTH GRAPHICS TOOLBOX (A USER’S GUIDE FOR USE WITH RFF FORTH)** |  |  |  |

**6. AUTHOR(S)** | **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** | **8. PERFORMING ORGANIZATION REPORT NUMBER** | **NA**
---|---|---|---
**Hanseok Ko** | **Naval Surface Warfare Center** | **NAVSWC MP 91-404** |  |
**10901 New Hampshire Avenue** |  |  |  |
**Silver Spring, MD 20903-5000** |  |  |  |

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** |  |  |
---|---|---|---
**11. SUPPLEMENTARY NOTES** |  |  |  |
**12a. DISTRIBUTION/AVAILABILITY STATEMENT** | **12b. DISTRIBUTION CODE** |  |  |
**Approved for public release; distribution is unlimited.** |  |  |  |

**13. ABSTRACT (Maximum 200 words)**

**FORTH GRAPHICS TOOLBOX** has a rich collection of graphics routines immediately useful for all FORTH based application software running on IBM-PC clone microcomputers. The routines are built upon graphics related primitives of both video BIOS call functions and direct-video functions. The user can develop more exotic application software based on the routines listed in this package.

**14. SUBJECT TERMS**

<table>
<thead>
<tr>
<th>Graphics</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routines</td>
<td>FORTH</td>
</tr>
<tr>
<td>Software</td>
<td>Underwater acoustics signal processing</td>
</tr>
</tbody>
</table>

**15. NUMBER OF PAGES**

65

**16. PRICE CODE**

SAR

**17. SECURITY CLASSIFICATION OF REPORT**

UNCLASSIFIED

**18. SECURITY CLASSIFICATION OF THIS PAGE**

UNCLASSIFIED

**19. SECURITY CLASSIFICATION OF ABSTRACT**

UNCLASSIFIED

**20. LIMITATION OF ABSTRACT**

UNCLASSIFIED

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
290.103
**GENERAL INSTRUCTIONS FOR COMPLETING SF 298**

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines** to meet optical scanning requirements.

<table>
<thead>
<tr>
<th>Block 1. Agency Use Only (Leave blank).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>C - Contract</th>
<th>PR - Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>G - Grant</td>
<td>TA - Task</td>
</tr>
<tr>
<td>PE - Program</td>
<td>WU - Work Unit</td>
</tr>
<tr>
<td>Element</td>
<td>Accession No.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 10. Sponsoring/Monitoring Agency Report Number. (If Known)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in...; When a report is revised, include a statement whether the new report supersedes or supplements the older report.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DOD - See DoDD 5230.24, &quot;Distribution Statements on Technical Documents.&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOE - See authorities.</td>
</tr>
<tr>
<td>NASA - See Handbook NHB 2200.2</td>
</tr>
<tr>
<td>NTIS - Leave blank.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 12b. Distribution Code.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DOD - Leave blank.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.</td>
</tr>
<tr>
<td>NASA - Leave blank.</td>
</tr>
<tr>
<td>NTIS - Leave blank.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 15. Number of Pages. Enter the total number of pages.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Block 16. Price Code. Enter appropriate price code (NTIS only)</th>
</tr>
</thead>
</table>

|----------------------------------------------------------------------------------------------------------------------------------|

<table>
<thead>
<tr>
<th>Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.</th>
</tr>
</thead>
</table>