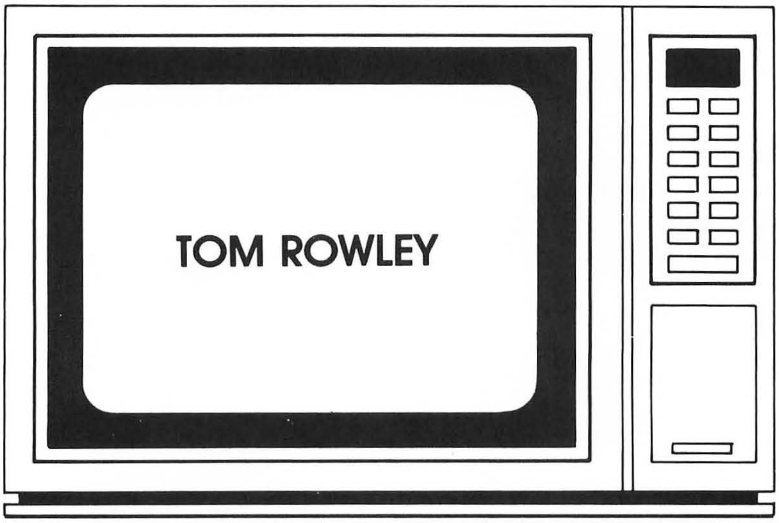
DESIGNS FROM YOUR MIND
with ATARI® GRAPHICS

Tom Rowley

DESIGNS FROM YOUR MIND
with ATARI® Graphics

DESIGNS FROM YOUR MIND with ATARI® Graphics



TOM ROWLEY

Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

Library of Congress Cataloging in Publication Data
Rowley, Tom.
Designs from your mind with ATARI graphics.

Bibliography: p.
Includes index.
1. Computer graphics. 2. Atari computer—Programming.
3. BASIC (Computer program language) I. Title.
T385.R68 1983 001.64'43 83-4560
ISBN 0-8359-1276-0

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America.

Contents

Acknowledgments

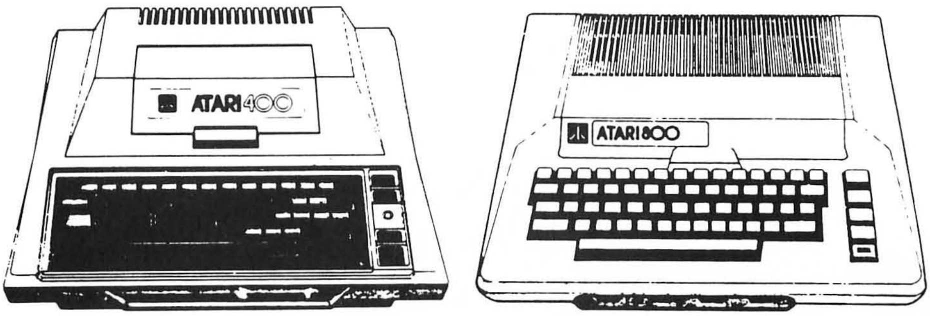
Thanks to Sam Calvin, Ron Leckrone, and Sebastian Michelena for their creative contributions to this book. A special thanks to Barbara McGlinchey for the loan of her creative talents. She was the inspiration behind many of the ideas presented in this book.

Introduction

Would you like to learn the secrets of an artist? This is your chance. However, the secrets you learn will not make you an artist in the typical sense of the word. You will become a graphics designer, a video composer, a computer artist.

This book is primarily about video graphics design. Graphics design of this sort is more than displaying a video image on a television screen. It is the art of electronic video imagery. It is shape, color, balance, feeling, and motion.

This book is also about communication. It is about instructing the computer to display your ideas on a TV screen, just as the artist expresses his ideas on canvas. Each artist needs a set of tools. The primary tool required for use with this book is an ATARI 800 or an ATARI 400 computer. It is the video artist's paint and brushes. Much of the information presented in this book involves color, therefore a color video screen is essential. It is the video artist's canvas.



Your creative ideas will be communicated to the computer and TV screen through the ATARI BASIC computer language. The computer must have an ATARI BASIC cartridge inserted into the appropriate slot.

A disk drive or a cassette recorder is not required but will be helpful in saving programs.

This book is a tutorial divided into two parts. Part One is an introduction to shapes, colors, and screen design. This part does not require an extensive computer background. It is for novices. A basic familiarity with the ATARI computer and its operation, however, is necessary and must be gained through sources such as the ATARI Owner's Manual. An elementary knowledge of the BASIC computer language would be helpful.

Advanced graphics techniques are examined in Part Two. These techniques are specifically related to ATARI graphics and the internal architecture of the ATARI computer. Although it is a continuation of Part One, a knowledge of BASIC programming is essential and a knowledge of assembler programming is desirable.

As you begin this book and use the computer and its programming language to explore computer graphics, keep these thoughts in mind:

- This book is about an exciting feeling from within.
- This book is about a stimulation of your visual sense.
- This book is about a sense of satisfaction.
- This book is about the opportunity to unleash your creative talents.
- This book is about your graphics creations—*Designs From Your Mind.*

PART ONE

CHAPTER 1

The Video Canvas

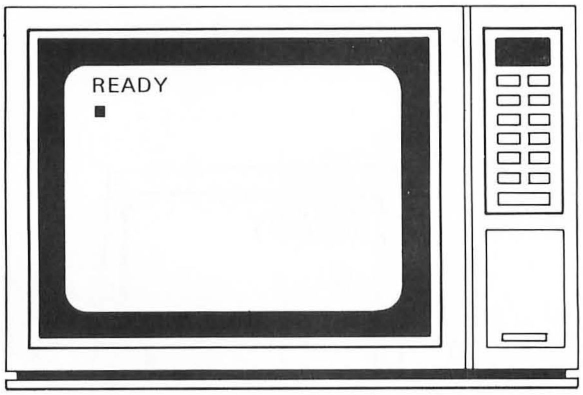
Are you ready?

This chapter is an introduction to ATARI graphics. It is our first exposure to the video canvas. In this chapter you will learn to:

- Change screen colors
- Print colored text
- Draw on the TV screen

Let's begin. Turn the computer on.

The word READY should appear in the upper left corner of the screen with the cursor (little white square) below it. Refer to the ATARI owner's manual if you need help.



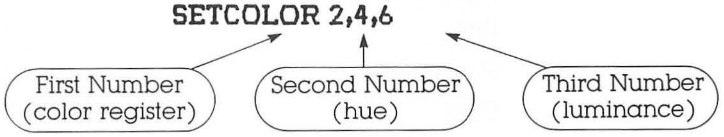
Changing Colors

Let's communicate with your computer. Type the following statement.

SETCOLOR 2,4,6 RETURN ← Press the **RETURN** key

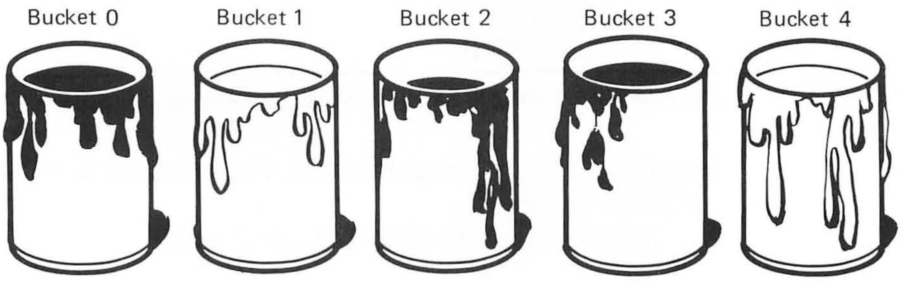
Did the screen turn pink? If it did, things are going smoothly. (*Note:* The color of some TV sets may vary or may need adjustment.) If you make a mistake, the computer will respond with an ERROR message. Don't be alarmed. Simply type the statement again.

The SETCOLOR statement is used to change or set colors to be displayed on the TV screen. It is a programming statement of the ATARI BASIC computer language. The SETCOLOR statement uses three numbers to set colors.



```
SETCOLOR 2,4,6
```

The *first number* is the color register. Your computer has five color registers which can be used with the SETCOLOR statement. You can think of each of these color registers as a place to store a particular color of paint, somewhat like a paint bucket.



We can put any color of paint into the buckets (color registers) that we like. The second and third numbers of the SETCOLOR statement will be used to determine the color of paint. We can use the paint from these buckets (registers) to change the colors of the background, border, words, and pictures on the TV screen.

The *second number* is the hue number. This number ranges from 0 to 15 and represents colors like red, blue, green, etc. The hues and their number equivalents are listed in Table 1-1.

The *third number* is the luminance number. It represents the brightness or darkness of the color. This number ranges

TABLE 1-1. Color Hue Values

Color	Hue Number (Second Number)
Gray	0
Light Orange	1
Orange	2
Red-Orange	3
Pink	4
Purple-Blue	6
Blue	7
Blue	8
Light Blue	9
Turquoise	10
Green-Blue	11
Green	12
Yellow-Green	13
Orange-Green	14
Light Orange	15

From the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

from 0 to 14 in increments of 2. The number 0 is the darkest and 14 is the brightest.

Do you understand? Let's find out.

What background color will the following statement make the TV screen?

```
SETCOLOR 2,13,4
```

(*Hint:* What color is represented by the number 13 in Table 1-1?)

Answer #1 ______________

(Answers to questions are printed upside down at the bottom of the page.)

#1. Yellow-Green

Change the background color. Type

SETCOLOR 2,13,4 RETURN

What statement can be used to change the TV screen to a light blue hue with a luminance of 4? Refer to Table 1-1.

Answer #2 ________________

Let's look at another color register. We'll use a 4 for the first number in the SETCOLOR statement. Type

SETCOLOR 4,2,6 RETURN

The border color should now be orange. You guessed it, color register 4 controls the border color.

Now change it to black. Type

SETCOLOR 4,0,0 RETURN

Can you change the border color to white. What statement will you type? (*Hint:* Use a hue of gray with a bright luminance.)

Answer #3 ________________

Are you ready for more excitement? Let's try color register 1. Type

SETCOLOR 1,9,14 RETURN

The words should be brighter. This color register controls the brightness (luminance) of the words on the screen. The second

#3. SETCOLOR 4,0,14
#2. SETCOLOR 2,9,4

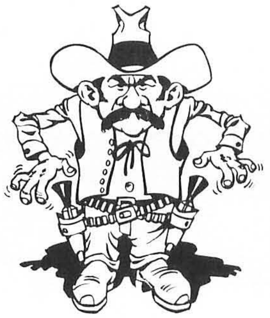
number in the SETCOLOR statement has no effect. The words are the same color as the background. For now, you can only change their luminance and not their color. Later we will color the words.

How could you make the words look as if they disappeared? Try making the words the same color as the background. This technique will be used later for hiding images on the screen.

Caution! Color register 2 does not always control the background color. Likewise, register 4 will not always control the border color and register 1 will not always change character luminance. Hang on until we study color registers in Chapter 2.

Throughout this book challenges will be issued. Some will be easy and some will be difficult. There may be many possible solutions to each challenge. One possible solution for each challenge is listed in Appendix B.

Here is our first challenge.



Challenge #1

Make the screen background color orange.

Make the border color blue.

Make the words as dark as possible.

The Graphics Screen

Ah, the excitement of color. But how do you print text and draw on the TV screen? To find out, read on.

Press the **SYSTEM RESET** key to return the screen to the original state. You can do this anytime you wish to start over.

Drawing and printing on the TV screen is not difficult. But first we must choose a GRAPHICS mode. The ATARI computer has a number of different GRAPHICS modes which are used to tell the computer the size of text and the resolution of drawings that you desire. Each of these modes uses a different type or format of graphics. Table 1-2 lists the GRAPHICS modes and the corresponding screen formats.

To understand the table, let's look at GRAPHICS mode 0. GRAPHICS mode 0 is a text mode. Text (characters) can be printed on the screen. The full screen is 40 columns wide and 24 rows high. (Split screens are studied later.) Two colors are possible in this mode. A more complete discussion of these GRAPHICS modes will take place in Chapters 2, 3, 4, and 5.

TABLE 1-2. GRAPHICS Modes and Screen Formats

		Screen Format		
GRAPHICS Mode	Mode Type	Columns	Rows full/split screen	Number of colors
0	Text	40	24/—	2
1	Text	20	24/20	5
2	Text	20	12/10	5
3	Graphics	40	24/20	4
4	Graphics	80	48/40	2
5	Graphics	80	48/40	4
6	Graphics	160	96/80	2
7	Graphics	160	96/80	4
8	Graphics	320	192/160	2
9	Graphics	80	192/—	16
10	Graphics	80	192/—	9
11	Graphics	80	192/—	16

Note: Modes 9, 10, and 11 will only work on computers which have a GTIA video chip.

From the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

Printing Text

Generally speaking, two types of screen displays are possible—text and graphics. For the text modes (GRAPHICS 0, 1, and 2), three different sizes of characters are possible. You have already seen the size of GRAPHICS mode 0 text. You have been doing your art thus far with a GRAPHICS mode 0 screen. Although the screen is now set to GRAPHICS mode 0, let's instruct the computer to use GRAPHICS 0 anyway. Type the following:

```
GRAPHICS 0 RETURN
```

You may notice that the GRAPHICS instruction causes the screen to clear. To print text on the screen you can use the PRINT statement. Type the following:

```
PRINT "GRAPHICS MODE 0" RETURN
```

The sentence in quotes is printed on the screen immediately below your statement. But, suppose you want to print the sentence elsewhere on the screen. To position this sentence, type the following.

```
POSITION 10,15:PRINT"GRAPHICS MODE 0" RETURN
```

We have placed two statements on the same line. First we set the position and then we print the text. The two statements are separated by a colon. The beginning print position is set to 10 characters from the left side of the screen and 15 lines from the top of the screen. We'll study this again in Chapter 5.

Now let's look at a GRAPHICS mode 1 screen. Type the following statement:

```
GRAPHICS 1 RETURN
```

Hummm? What happened? The screen has become split. The top portion of the screen is GRAPHICS 1 and bottom blue portion remains GRAPHICS 0. The bottom portion is sometimes refered to as a *text window*. Type the following statement:

PRINT#6;"GRAPHICS 1"

And there you have it—two sizes of text on the same screen. PRINT#6; instructs the computer to print on the top portion of a split text screen. The sentence in quotes is printed just as before. Notice that GRAPHICS mode 1 produces characters that are double the width of those in GRAPHICS mode 0. Believe it or not the height is the same. Also notice that the background and character colors have changed. We'll set the colors of them later.

Ok, we've gone this far. We might as well see what GRAPHICS mode 2 characters look like. Type the following statement:

GRAPHICS 2 RETURN

Poof! The screen should have cleared. And now type:

PRINT#6;"GRAPHICS 2" RETURN

And now you see it. GRAPHICS mode 2 characters are twice as high and twice as wide as those of GRAPHICS mode 0.

?

What two statements would you use to print THE END in double-width, single-height characters?

Answer #4 ____________________
____

Later we'll show how to mix all of these GRAPHICS modes on a single screen. But for now, let's look at another GRAPHICS mode.

#4. GRAPHICS 1
PRINT#6;"THE END"

Drawing Objects on the Screen

We might as well start with GRAPHICS mode 3. You guessed it, type

GRAPHICS 3 RETURN

Again the screen is cleared.

Normally, it is not possible to print text on a GRAPHICS screen. It is possible, however, to plot points on the screen. To plot a point on the screen you need to do two things. First, you must instruct the computer as to which color you desire to use and, second, you must instruct the computer where to plot the point.

The first part is done with the COLOR statement. In this case, let's choose the color numbered 1. We will take a detailed look at the COLOR statement later. Type the following statement:

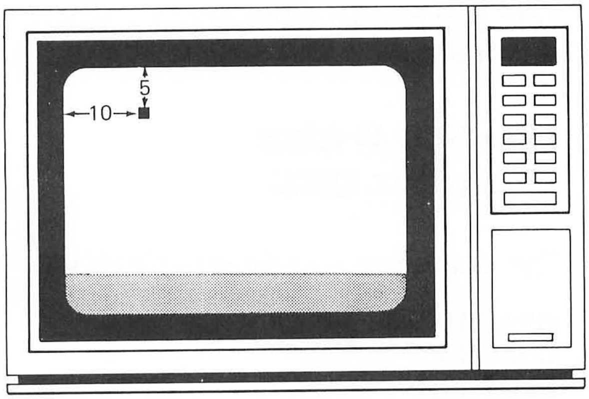
COLOR 1 RETURN

You're right. You can see no effect. But wait until you PLOT. Try this. Type

PLOT 10,5 RETURN

You have plotted an orange square. The square is orange because the color register used by the COLOR 1 statement normally contains orange paint.

Let's take a closer look at the PLOT statement. The top portion of the GRAPHICS mode 3 split screen is divided into 40 squares horizontally and 20 squares vertically. Refer to Table 1-2 and the GRAPHICS mode 3 worksheet in Appendix A. The coordinate 10,5 is located in the upper left portion of the screen. This means that there are 10 empty squares to the left and 5 empty squares above the coordinate 10,5. The PLOT statement is similar to the POSITION statement. The POSITION statement sets a place on the screen to print text whereas the PLOT statement actually marks the spot with a color.



Where on the screen is the point 30,5 located? (Upper right, upper left, lower right, lower left?)

Answer #5 ________________

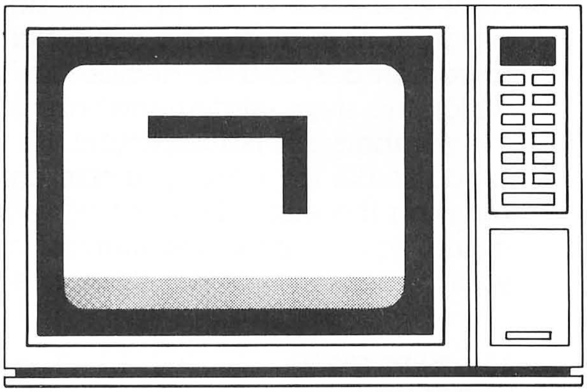
We can draw a line (a very fat one) by typing the following statement. Try it.

DRAWTO 30,5 RETURN

Slick, huh? And now type

DRAWTO 30,15 RETURN

It should look like this



#5. Upper right

Let's complete the rectangle. It will require two more DRAWTO statements.

```
DRAWTO 10,15 RETURN
DRAWTO 10,5 RETURN
```

The entire series of statements would look like this.

```
GRAPHICS 3
COLOR 1
PLOT 10,5
DRAWTO 30,5
DRAWTO 30,15
DRAWTO 10,15
DRAWTO 10,5
```

This series of steps constitutes a computer program. It is a computer program written in ATARI BASIC to draw a rectangle in GRAPHICS mode 3. Thus far, each instruction statement we've given the computer is done immediately. But the computer does not remember the set of instructions for the rectangle.

Follow this procedure to make the computer remember the entire sequence of statements:

1. Press **SYSTEM RESET**
2. Type NEW **RETURN**. This statement will erase any program (set of statements) that was previously in the computer.
3. Now type each of the program statements with a line number in front of them. The computer uses these line numbers to remember and to decide on the correct sequence in which the steps (statements) are to be done. The values of the numbers are not important. It's the sequence that counts. If you make an error, you can change any line by simply retyping the entire line as you wish it to appear. To delete a line, type only the line number and press **RETURN**. Refer to your owner's manual for assistance in making corrections.

```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 10,5
```

Remember, Press **RETURN** after each line.

```
40 DRAWTO 30,5
50 DRAWTO 30,15
60 DRAWTO 10,15
70 DRAWTO 10,5
```

4. To see if you have typed it correctly, type

   LIST RETURN

   It should be listed on the screen as it appears above. If not, correct it.

5. Ok, are you ready for the big moment? Type

   RUN RETURN

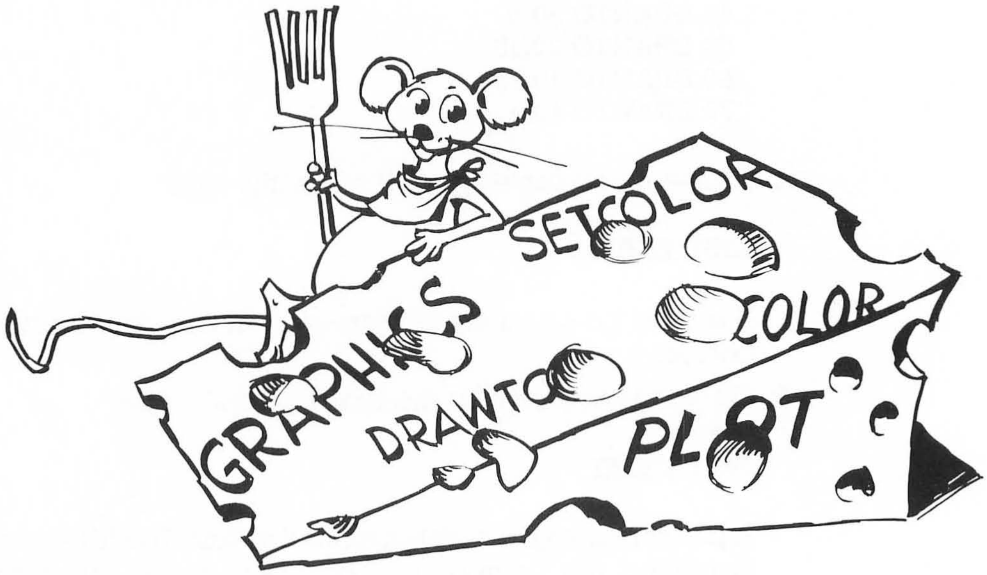
   A pretty rectangle, it all happened so fast. The RUN command instructed the computer to do each of the program statements in the order of the line number. The computer remembers all of the program statements but did not use them until we used the RUN command.

We could make a similar rectangle in GRAPHICS modes 4 through 11. They are similar to mode 3 in the way they are used. We will study them in Chapter 2.

Challenge #2

Write a program to draw a square in GRAPHICS mode 3. Each side should be 10 units long. The upper left hand corner should be plotted at position 10,5.

You've learned a lot but, as this chapter ends, your mind probably feels like a piece of swiss cheese. There seems to be a lot of holes in what you've learned.



The next five chapters will be spent on an in-depth study of the Chapter 1 exposure. But, before we go on, let's do the Chapter 1 Review.

Chapter 1 Review

You used the following ATARI BASIC program statements in this chapter.

SETCOLOR GRAPHICS COLOR PRINT
PLOT DRAWTO POSITION

Can you remember what each statement does? Match each of the above GRAPHICS statements with the following descriptions.

1. ______________ clears the screen and sets the screen format.
2. ______________ changes the values in the color registers. Can be used to change background, border, and text colors.

3. ______________ displays a point (square) on the screen.
4. ______________ is used to choose the color of points and lines to be drawn on the screen.
5. ______________ is used to connect two points with a line.
6. ______________ is used to specify a place on the screen to begin printing.
7. ______________ causes text to be placed on the screen in GRAPHICS modes 0,1, and 2.

The following statements will change the background color of the screen in GRAPHICS mode 0. What color results from using each statement?

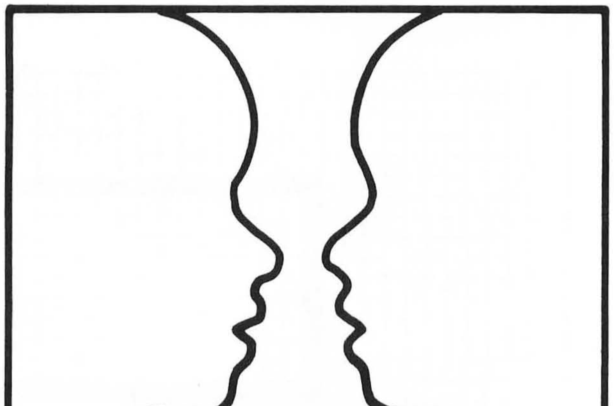
8. SETCOLOR 2,4,8 ______________
9. SETCOLOR 2,12,8 ______________
10. SETCOLOR 2,2,8 ______________
11. The following program is used to draw an image on the TV screen. Sketch the shape on a GRAPHICS mode 3 worksheet.

```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 2,10
40 DRAWTO 2,15
50 DRAWTO 37,15
60 DRAWTO 37,10
```

CHAPTER 2

Shapes, Colors, and GRAPHICS Modes

Shape is one of the most basic perceptions of the optical sense. We identify objects by their shape. You can identify the letters on this page by their shape. We also attach meanings to certain shapes and combinations of shapes. What do you perceive this shape to represent?



It could be a vase or perhaps two people talking.

It is the shape displayed on the TV screen that carries the largest amount of information. Although colors enhance and clarify, it is this ability to draw that is most important to the video composer.

In this chapter we will investigate the ATARI BASIC GRAPHICS modes. You will learn to:

- Draw shapes of different sizes
- Change the color of shapes
- Use GRAPHICS modes 3 through 8

Shapes and the Graphics Screen

In Chapter 1 we drew a shape. It was a rectangle drawn in GRAPHICS mode 3. The program for a rectangle was fairly easy.

Figure 2-1 is a layout of the rectangle on a mode 3 worksheet. The screen design worksheets in Appendix A should be used to assist you in laying out your design.

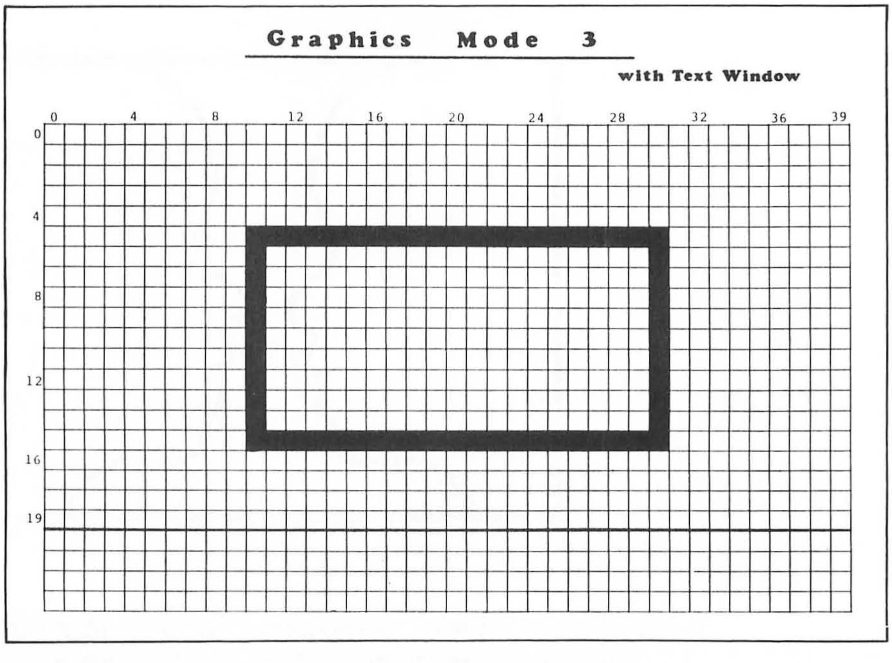


FIGURE 2-1. Layout of Rectangle on Mode 3 Worksheet.

Recall, from Chapter 1, the program statements for the above rectangle.

```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 10,5
40 DRAWTO 30,5
50 DRAWTO 30,15
60 DRAWTO 10,15
70 DRAWTO 10,5
```

The upper left hand corner of the screen has the coordinate 0,0. The upper right hand corner has the coordinate 39,0. This means that the horizontal coordinates range from 0 to 39 and provide a screen format that is 40 positions wide.

This same reasoning applies to the vertical format. Although the screen can have a total of 24 vertical positions, the bottom position on the left side of the screen has a coordinate of 0,23. Again, this is because 0 is counted as a position.

So, remember this: The minimum value in any coordinate is 0. The maximum value is one less than the total number of positions.

Another point should be made concerning the worksheet. GRAPHICS mode 3, like most other GRAPHICS modes, can be used in either a split screen or a full screen format. Thus far, we have only used the split screen format. Later in this chapter we will use the full screen format.

In the split screen format, the screen is divided into two parts. The top portion of the screen could be any GRAPHICS mode 1 through 8. The bottom portion of the screen is GRAPHICS mode 0. This bottom portion, or text window, can hold four lines of GRAPHICS mode 0 text. Later in this book we will learn to place text windows at different positions in the graphics screen.

Challenge #2 was to draw a square. By adjusting the coordinates, squares of different sizes could be drawn anywhere on the screen. This is your challenge.

Challenge #3

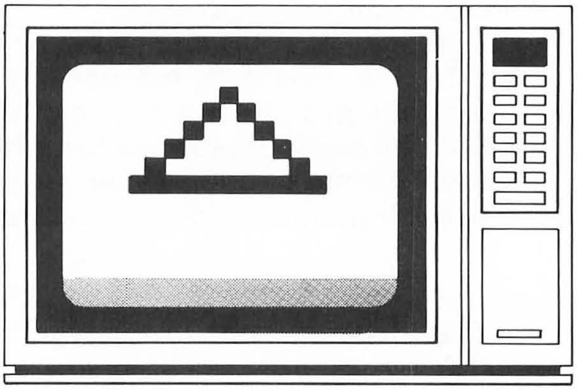
Write the set of program statements to draw a square six units on a side in the center of the top part of the split screen in GRAPHICS mode 3.

Let's choose another shape to draw. Here is a program that draws a triangle:

```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 20,5
40 DRAWTO 30,15
50 DRAWTO 10,15
60 DRAWTO 20,5
```

Type NEW to clear any old programs. Enter each program line. We will assume for the reminder of the book that the **RETURN** key is pressed each time a program line is entered.

Run the program. The screen should look like this:



Do you understand the GRAPHICS mode 3 screen?

Which program lines could you change to make the base of the triangle a different size?

Answer #1 ________________

Let's make the triangle a different size. Change these two lines.

```
40 DRAWTO 35,15
50 DRAWTO 5,15
```

Run the program again. Does the triangle look different? Now, answer another question.

What two statements would you use to draw a diagonal line from the upper right hand corner to the lower left hand corner of the screen in GRAPHICS mode 3?

Answer #2 ________________
____

About GRAPHICS Modes

The ATARI computer supports a number of different GRAPHICS modes. We will look at modes 3 through 8 here and delay our study of the other modes until later chapters. The differences in these modes lie primarily in the varying resolutions and colors available to the user.

#1. Lines 40 and 50

#2. PLOT 39,0
DRAWTO 0,19

Let's do some simple experiments. To begin, clear the screen by pressing the **SYSTEM RESET** key. Erase any old programs in the computer by using the NEW statement. Remember, this needs to be done whenever you start a new program.

Enter the following program.

```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 20,10
```

Run the program. The square that appears on the screen is called a *pixel.* The size of this pixel will change for the various GRAPHICS modes.

Try the above program with line 10 changed to GRAPHICS 4. You only need to retype line 10. Your program should look as shown. List it with the LIST command to see if it's the same.

```
10 GRAPHICS 4
20 COLOR 1
30 PLOT 20,10
```

Run the program. The pixel size is smaller than in mode 3. Try the program in GRAPHICS modes 5, 6, 7, and 8 to see the size of the pixels.

?

Which GRAPHICS mode has the highest resolution (smallest pixel size)?

Answer #3 ________________

You should note that GRAPHICS modes 4 and 5 have the same size pixels and modes 6 and 7 have the same size pixels. We will see later that they are different in the number of colors that can be displayed. The background changed color in GRAPHICS 8 and a very small pixel was located at the upper

#3. GRAPHICS Mode 8

left hand corner of the screen. Now, let's go back to the triangle that we previously drew and see what it looks like in GRAPHICS mode 8. Try the following program:

```
10 GRAPHICS 8
20 COLOR 1
30 PLOT 20,5
40 DRAWTO 30,15
50 DRAWTO 10,15
60 DRAWTO 20,5
```

The triangle appears to be much more refined than it was in GRAPHICS mode 3. But, also notice that the position of the triangle on the screen has changed and it is considerably smaller. Because GRAPHICS 8 pixels are smaller than GRAPHICS 3 pixels, the triangle is smaller and closer to the upper left hand corner of the screen.

Building a House

Ok, it's time for a project. We are going to draw a very simple little house. The top part of the house will be made of a triangle and the bottom part will be made of a square. How convenient, we have already drawn both of these shapes.

Let's select GRAPHICS mode 5 for our drawing. Get out a mode 5 worksheet from Appendix A. It would be a good idea to make copies of the screen design worksheets. Lay out your design in the center portion of the worksheet. It might look something like Figure 2-2.

The program for the house includes the statements for a triangle and the statements for a square.

```
210 GRAPHICS 5
220 COLOR 1
310 PLOT 40,5
320 DRAWTO 50,15
330 DRAWTO 30,15
340 DRAWTO 40,5
```

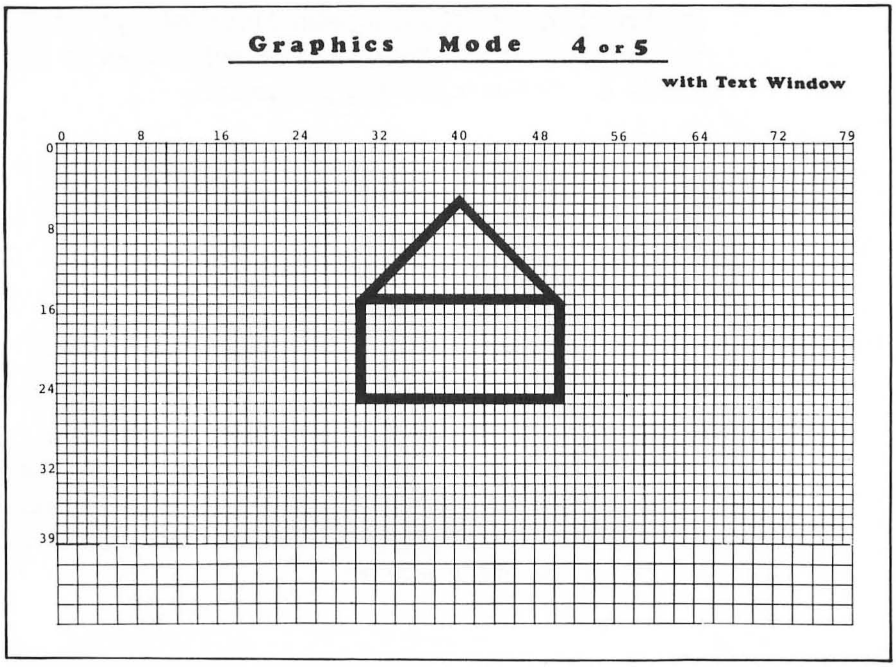


FIGURE 2-2. Layout of House on Mode 5 Worksheet.

```
410 PLOT 30,15
420 DRAWTO 50,15
430 DRAWTO 50,25
440 DRAWTO 30,25
450 DRAWTO 30,15
```

Enter the lines and run the program. Does the house look as you predicted?

Before we continue, let's take time out to document what we've done.

An important concept in computer programming is to organize and document your work. You can then save your work and return at a later time and your documentation will help you remember what you've done. It will also be easier for others to read your work.

We could document the above program by adding four REM (remark) statements. These statements have no effect on the operation of the program. They may be placed on separate lines or on lines with other statements. If you place them on lines with other statements, you need to place a colon in front of the REM statement.

Add the following lines to your program. Just type them as shown.

YOUR NAME and DATE go here

```
100 REM A LITTLE HOUSE BY TOM ON 11 DECEMBER 1982
200 REM SET UP GRAPHICS SCREEN
300 REM DRAW THE TOP OF THE HOUSE
400 REM DRAW THE BOTTOM OF THE HOUSE
```

Each program should also have an END statement. This statement tells the computer that the program is finished. Add the END statement to your program.

```
500 END
```

Run the program again. It should work the same as before.

Changing the House Color

Let's get back to our house. Suppose that you don't want a yellow house. Suppose that you want to paint your house a bright pink. Let's try it. Add this line to your program:

```
230 SETCOLOR 0,4,10
```

Run the program. Now you have a bright pink house.

But this is all so confusing. How do you know that SETCOLOR 0,4,10 changes the house color? Here is an explanation. The COLOR and SETCOLOR statements work together when you plot or draw on the screen. The COLOR statement specifies a color register to be used when drawing on the screen. The SETCOLOR statement is used to set or change the color in the register.

It can seem a little complicated so let's look at the colors we used in our house program. If we list our house program, it looks like this.

```
100 REM A LITTLE HOUSE BY TOM ON 11 DECEMBER 1982
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 5
220 COLOR 1
230 SETCOLOR 0,4,10
300 REM DRAW THE TOP OF THE HOUSE
310 PLOT 40,5
320 DRAWTO 50,15
330 DRAWTO 30,15
340 DRAWTO 40,5
400 REM DRAW THE BOTTOM OF THE HOUSE
410 PLOT 30,15
420 DRAWTO 50,15
430 DRAWTO 50,25
440 DRAWTO 30,25
450 DRAWTO 30,15
500 END
```

Note that three lines determine the house color. They are

```
210 GRAPHICS 5
220 COLOR 1
230 SETCOLOR 0,4,10
```

Table 2-1 describes the relationship between the GRAPHICS, COLOR, and SETCOLOR statements.

First look at the table and find GRAPHICS mode 5. Since we used the statement COLOR 1 in our program, find the number 1 under the column titled COLOR and in the same row as GRAPHICS mode 5. Notice that the entire line for this example has been shaded on Table 2-1. By reading the row, you should notice that for GRAPHICS mode 5, COLOR 1 goes together with SETCOLOR for register 0. In our house program, we changed the house to pink with the statement SETCOLOR 0,4,10.

Let's change the house to a blue color. Change line 230 to

```
230 SETCOLOR 0,8,6
```

Run the program again. Is the house blue? Again, to change COLOR 1 in GRAPHICS 5 we have used SETCOLOR for register 0.

TABLE 2-1. GRAPHICS MODE, SETCOLOR, COLOR

GRAPHICS Mode	COLOR	SETCOLOR Register #	Comments
0	not used	0	— not used
		1	character luminance
		2	background
		3	— not used
		4	border
1 and 2	not used	0	character
		1	character (text window character luminance)
		2	character (text window back-ground)
		3	character
		4	background, border
3, 5, and 7	1	0	graphics point
	2	1	graphics point (text window character luminance)
	3	2	graphics point (text window background)
	—	3	— not used
	0	4	graphics point (background, border)
4 and 6	1	0	graphics point
	—	1	(text window character luminance)
	—	2	(text window background)
	—	3	— not used
	0	4	graphics point (background border)
8	—	0	— not used
	1	1	graphics point luminance
	0	2	graphics point (background)
	—	3	— not used
	—	4	border

Adapted from the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

But why was the house orange to begin with? Well, color register 0 is normally set to orange when you turn the computer on. In fact, each of the color registers are set with a default color when the computer is turned on. These colors are listed in Table 2-2.

TABLE 2-2. Color Register Default Values

SETCOLOR	Defaults to		
Register #	Hue	Luminance	Actual Color
0	2	8	Orange
1	12	10	Green
2	9	4	Dark Blue
3	4	6	Pink
4	0	0	Black

Let's go back to our house program and see if you understand. Change line 220 to

```
220 COLOR 2
```

Run the program. The house is green because that is the default color when we draw with COLOR 2 in GRAPHICS mode 5.

To set the house color we now must use color register 1. Refer to Table 2-1. Change line 230 to

```
230 SETCOLOR 1,0,14
```

Run the program. Your house should be white.

If we change line 220 to 220 COLOR 3 what color will the house be?

Answer #4 ________________

#4. Dark blue

What SETCOLOR statement must be used to make the house white when drawing with COLOR 3?

Answer #5 ________________

Let's use two different colors and make the bottom of the house a different color from the top. Change your program so that it appears as follows.

```
100 REM A LITTLE HOUSE BY TOM ON 11 DECEMBER 1982
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 5
300 REM DRAW THE TOP OF THE HOUSE
305 COLOR 1
310 PLOT 40,5
320 DRAWTO 50,15
330 DRAWTO 30,15
340 DRAWTO 40,5
400 REM DRAW THE BOTTOM OF THE HOUSE
405 COLOR 2
410 PLOT 30,15
420 DRAWTO 50,15
430 DRAWTO 50,25
440 DRAWTO 30,25
450 DRAWTO 30,15
500 END
```

Run the program. The top should be orange (default color of register 0) and the bottom should be green (default color of register 1). The COLOR 1 statement is used for lines 310 through 340. The COLOR 2 statement is used for lines 410 through 450.

Let's change another color register. Add the following line:

240 SETCOLOR 4,4,4

Can you predict what will happen? Run the program and see if you're right.

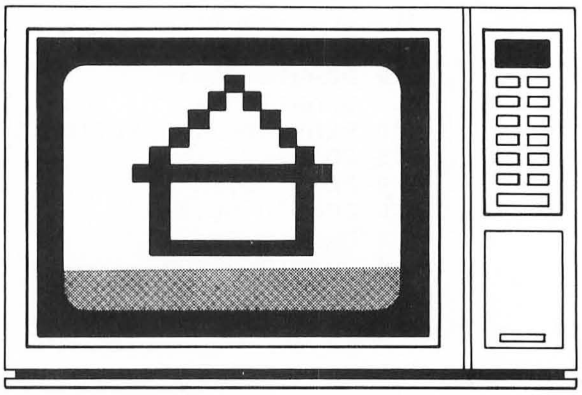
#5. SETCOLOR 2,0,14

What statement would you use to change the background color to blue?

Answer #6 ________________

Challenge #4

Draw a house in GRAPHICS mode 7. It might look something like the following picture. You choose the size. Make the background black, the top red, and the bottom blue.



Areas of Color

There are a couple of ways to fill areas of the screen with color. One way is to continually draw adjacent lines until the desired area is filled. We can try a sample program. Erase any old programs and enter the following lines.

#6. SETCOLOR 4,8,4 (Answers may vary)

```
5 REM COLOR AREA PROGRAM
100 REM COLOR AREA PROGRAM
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 COLOR 1
300 REM DRAW 3 LINES
310 PLOT 10,20
320 DRAWTO 150,20
330 PLOT 10,21
340 DRAWTO 150,21
350 PLOT 10,22
360 DRAWTO 150,22
400 END
```

Run it. It seems like a lot of work to only draw three lines, huh?

Let's use a FOR . . . TO, NEXT loop. If you are unfamiliar with loops, you may want to refer to a BASIC programming text or the ATARI BASIC reference manual. Change the program to look like this:

```
5 REM COLOR AREA PROGRAM
100 REM COLOR AREA PROGRAM
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 COLOR 1
300 REM DRAW 3 LINES
310 FOR X=20 TO 22
320 PLOT 10,X
330 DRAWTO 150,X
340 NEXT X
400 END
```

Run it. The results should be the same as before.

In this program, X is a variable that begins with the value 20. As soon as the statement NEXT X is encountered, control returns to the FOR statement and X increases by 1 to 21. This continues until X reaches 22 at which time the loop ends. Notice that the value of X is also used in the PLOT and DRAWTO statements.

Now it's easy to draw a lot of lines. Go back and change line 310 to

```
310 FOR X=20 TO 60
```

Run it. You have drawn 41 adjacent lines. Later we will look at another way to fill areas of the screen with color.

Full Screen Graphics

It's time to get rid of the text window and use full screen graphics. This can be done by simply adding the number 16 to the GRAPHICS mode number.

Let's continue to use our last program. Line 210 could be changed to either

```
210 GRAPHICS 7+16
```

or

```
210 GRAPHICS 23
```

Change it and run the program. Oops! It disappeared when the drawing was completed.

Programs that come to a normal end return the screen to GRAPHICS mode 0. One way to prevent this is to tie the program up into an infinite loop. Adding line 390 will do this.

```
390 GOTO 390
```

When line 390 is reached, the program statement infinitely instructs the computer to go to line 390. Run the program. Notice how nice and clean the screen looks. You can break from this loop by pressing the **BREAK** key or the **SYSTEM RESET** key.

A lot of new information was presented in Chapter 2. Do the Chapter 2 Review to see how well you've done.

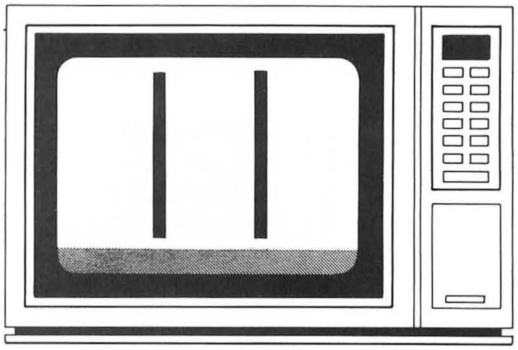
Chapter 2 Review

In this chapter you studied and experimented with GRAPHICS modes 3 through 8. Here are some questions to see how well you understand what you've studied.

DRAWINGS—Match each of the drawings with the correct program.

1. ________

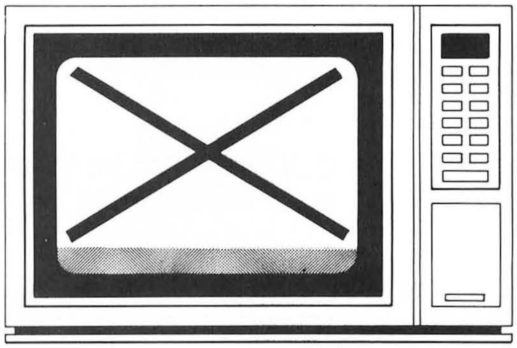
```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 0,0
40 DRAWTO 159,79
50 PLOT 0,79
60 DRAWTO 159,0
```



A

2. ________

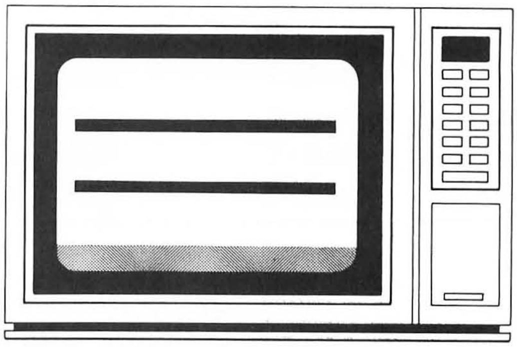
```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 53,0
40 DRAWTO 53,79
50 PLOT 106,0
60 DRAWTO 106,79
```



B

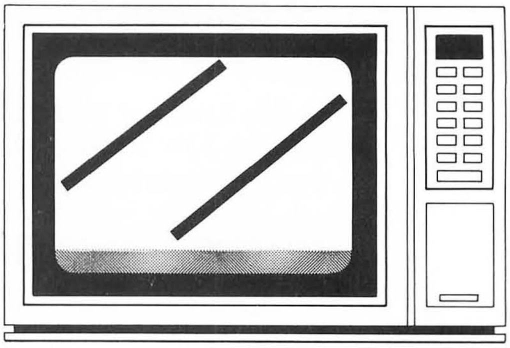
3. ________

```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 0,39
40 DRAWTO 79,79
50 PLOT 79,0
60 DRAWTO 159,39
```

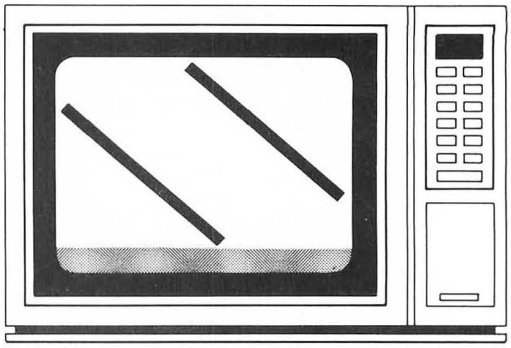


C

4. ________
```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 79,0
40 DRAWTO 0,39
50 PLOT 159,39
60 DRAWTO 79,79
```


D

5. ________
```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 0,27
40 DRAWTO 159,27
50 PLOT 0,54
60 DRAWTO 159,54
```

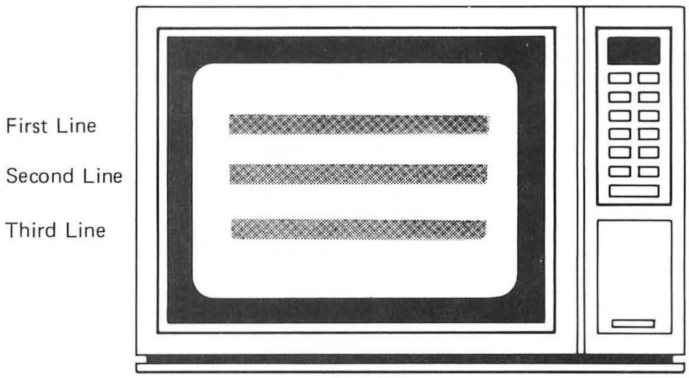

E

COLORS—For each of the following situations, specify which color register must be set.

	GRAPHICS Mode	To Set Color Condition	Color Register (SETCOLOR)
6.	Mode 5	Background color	________
7.	Mode 0	Border color	________
8.	Mode 4	COLOR 1 drawings	________
9.	Mode 1	Background color	________
10.	Mode 7	COLOR 3 drawings	________
11.	Mode 8	Background color	________
12.	Mode 3	COLOR 0 drawings	________

The following program is used to draw three lines on the TV screen as shown. What will be the color of each of the lines?

```
100 REM DRAW THREE COLORED LINES
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3
300 REM DRAW FIRST LINE
310 COLOR 1
320 PLOT 5,5
330 DRAWTO 35,5
400 REM DRAW SECOND LINE
410 SETCOLOR 1,8,4
420 COLOR 2
430 PLOT 5,10
440 DRAWTO 35,10
500 REM DRAW THIRD LINE
510 SETCOLOR 2,4,10
520 COLOR 3
530 PLOT 5,15
540 DRAWTO 35,15
600 END
```



13. Color of first line ____________________

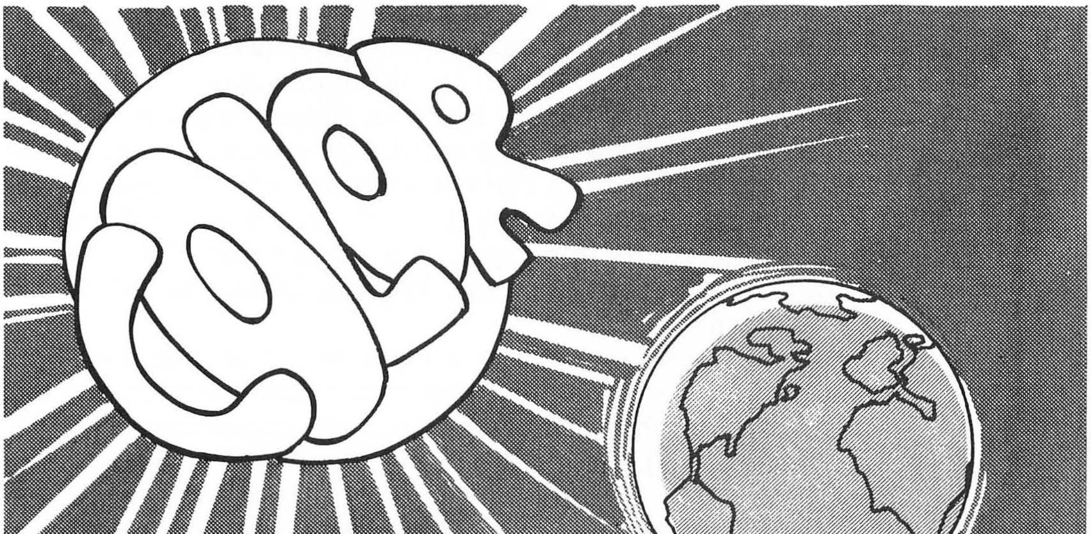
14. Color of second line ____________________

15. Color of third line ____________________

CHAPTER 3

Colors and Contrasts

Color! It is the sunshine of the world.



In Chapter 2 we learned to control the colors displayed on the TV screen. But color video graphics is much more than a display of varying screen colors. Just as the artist can express himself with colors on a canvas, so must the video composer be able to express a variety of feelings with the colors on the TV screen.

In this chapter we will continue to learn and use the GRAPHICS modes. Our primary objective, however, will be to gain an understanding of the art of color as it is specifically related to TV graphics.

In this chapter you will learn to:

- Select colors with appropriate contrast
- Make drawings with varying contrasts
- Use GRAPHICS modes 9, 10, and 11

Before beginning, we need to have a basic understanding of color.

The colors we see with our eyes are a direct result of the light present around us. In the absence of light (a dark room, for example) all appears black. In the presence of light, we see a multitude of color. Light (white light, specifically) is a combination of all colors of light. An object appears to have color because it reflects only a part (a specific color) of the white light that falls on it. For example, a red car appears red on a sunny day because only the red component of the white sunlight is reflected from the paint on the car. The red car absorbs the other components (yellow, blue, green) of the white sunlight.

In this chapter we will be working with colors as if we had paints and a brush. The only difference is that we will paint the TV screen rather than a canvas. In our experimentation, we will discuss the use of primary and secondary colors. A definition of these colors will be provided here. This definition concerns paints or pigments but will be of use in our study of color TV graphics. This definition is not to be confused with the colors of light and their properties. The primary and secondary colors are depicted in Figure 3-1.

The diagram shows the results of adding various colors of paints. For example, the mixing of the two primary colors yellow and blue produces the secondary color green. A secondary color results when any two primary colors are

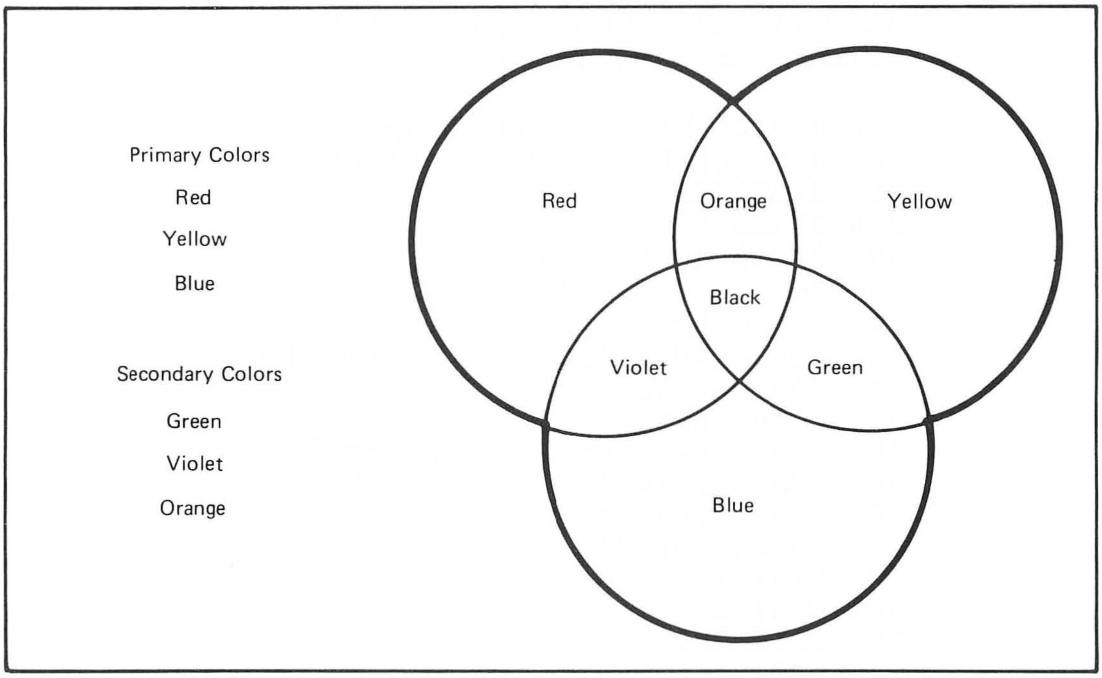


FIGURE 3-1. Primary and Secondary Colors.

mixed. The combination of all three primary colors or all three secondary colors produces black.

Contrasts

Many kinds of contrasts or comparisons exist: young-old, big-small, heavy-light, and so on. Colors may also have contrast. They may be light or dark. But colors may also be used to express relationships such as cold and warm, far and near, and wet and dry. Some colors may appear washed out when contrasted while others may appear sharp and brilliant. Contrasts of color truly provide a means of expressing a variety of feelings.

Are you ready to paint some contrasting colors? Let's write a color display program to be used in our experimentation. We will use this program throughout this chapter to explore color contrasts. Enter this program:

```
100 REM COLOR CONTRAST DISPLAY
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
220 SETCOLOR 0,4,4
230 SETCOLOR 1,1,14
240 SETCOLOR 2,8,4
250 SETCOLOR 4,0,14
300 REM DRAW 3 BLOCK IN COLORS 1,2,3
310 FOR C=0 TO 2
320 COLOR C+1
330 REM DRAW 7 HORIZONTAL LINES
340 FOR X=1 TO 7
350 PLOT 2,C*7+X
360 DRAWTO 38,C*7+X
370 NEXT X
380 NEXT C
390 GOTO 390
400 END
```

This program will draw three horizontal blocks of color on the screen. Each block will contain seven GRAPHICS mode 3 lines. Each of the SETCOLOR statements in lines 220 through 250 specifies an appropriate color. We will change these lines to display different colors on the screen.

?

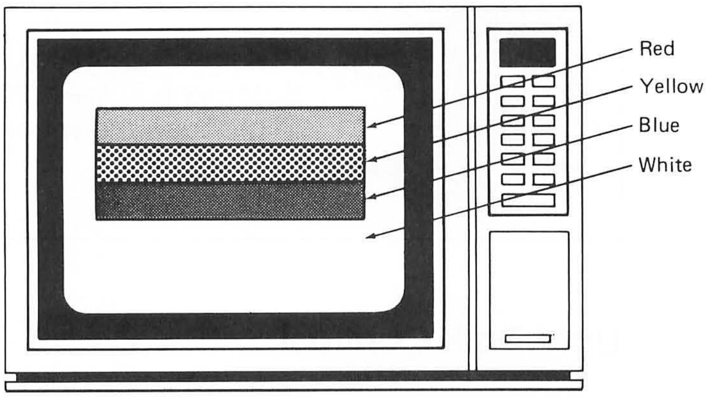
Refer to Table 1-1 and see if you can predict the colors that will be displayed. Fill in the blanks with the colors.

SETCOLOR 0,4,4 Answer #1 ____________________
SETCOLOR 1,1,14 Answer #2 ____________________
SETCOLOR 2,8,4 Answer #3 ____________________
SETCOLOR 4,0,14 Answer #4 ____________________

Notice that color register 3 is not used in GRAPHICS mode 3.

#1. Red #2. Yellow #3. Blue #4. White

Run the program. It should produce a display that looks something like this:



If the colors on your TV are not as indicated in the above diagram, this would be a good time to adjust your TV.

Contrast of Hue

The primary colors, red, yellow, and blue are the strongest contrasting hues. The contrast becomes sharper when the colors are separated by black or white lines.

Let's try it. Change line 340 to

```
340 FOR X=1 TO 6
```

Run it. Do you see a difference in contrast?

Change the background to black. Change line 250 to

```
250 SETCOLOR 4,0,0
```

Run it. Do you see a different contrast?

Here it comes, Challenge #5.

Challenge #5

Change the values of the SETCOLOR statements in the color display program to produce the secondary hues.

Light-Dark Contrast

Light and dark are the most fundamental of contrasts. The strongest contrasts of light and dark are the colors white and black. Light-dark contrast is relative. Gray may appear light compared to black but dark compared to white. Colors of equal luminance also provide light-dark contrast. The colors violet and yellow produce the strongest light-dark contrast. Let's change the SETCOLOR statements in the color display program to produce yellow and violet colors on the screen. Try the following changes.

```
220 SETCOLOR 0,6,14
240 SETCOLOR 2,6,14
```

Your program should look like this:

```
100 REM COLOR CONTRAST DISPLAY
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
220 SETCOLOR 0,6,14
230 SETCOLOR 1,1,14
240 SETCOLOR 2,6,14
250 SETCOLOR 4,0,0
300 REM DRAW 3 BLOCK IN COLORS 1,2,3
310 FOR C=0 TO 2
320 COLOR C+1
330 REM DRAW 7 HORIZONTAL LINES
```

```
340 FOR X=1 TO 6
350 PLOT 2,C*7+X
360 DRAWTO 38,C*7+X
370 NEXT X
380 NEXT C
390 GOTO 390
400 END
```

Run it. Notice the contrast between colors of equal luminance.

Cold-Warm Contrast

Color contrasts can be used to convey feelings of cold and warm. These same contrasts also imply feelings for other relationships. Some of these are listed.

cold—warm
shadow—sun
transparent—opaque
sedative—stimulant
rare—dense
air—earth
far—near
light—heavy
wet—dry

These relationships will be of use in Chapter 6 when we work with screen design.

The colors that provide the strongest cold-warm contrast are blue-green—red-orange. Other colors fall somewhere in between.

Let's modify our color display program to illustrate the cold-warm effect. Change these lines:

```
220 SETCOLOR 0,3,8
230 SETCOLOR 1,6,8
240 SETCOLOR 2,11,8
340 FOR X=1 TO 7
```

Run the program. Notice how violet appears warm compared to the blue green but cold compared to the red orange. The blue green feels far whereas the red orange seems close.

Suppose that you display the two colors brown and green. Which of these colors would convey the feeling of heavy in a light—heavy contrast?

Answer #5 ________________

Let's try another question. Suppose that you want to use color to illustrate a wet—dry relationship. What two colors might you choose?

Answer #6 ________________—________________

Complementary Contrast

Two colors are complementary if their pigments when mixed together produce a neutral gray-black. Refer to Figure 3-1 and note that when a primary color is mixed with the opposite secondary color, black is produced. These are called complementary pairs. Examples of complementary pairs are:

yellow, violet

blue, orange

red, green

Yellow/violet represents not only a complementary contrast but also an extreme light-dark contrast. This was illustrated by the last program.

Let's see what the blue, orange complementary pair looks like. Change line 220, 230, and 240 of our color display program.

#5. Brown #6. Blue-Orange (Answers may vary)

```
220 SETCOLOR 0,8,6
230 SETCOLOR 1,2,6
240 SETCOLOR 2,8,6
```

Run the program. Do you see the vivid contrast?

Simultaneous Contrast

The human eye becomes fatigued to a particular color after it has been exposed to it for a period of time. For example, if you were to look at the color yellow for a period of time, your eye would not be as responsive to yellow. You would not see yellow with the same vividness.

If you were to suddenly look at a neutral gray color, your eye would be more responsive to the complement or opposite of yellow (violet) and the gray color would appear to contain a tinge of violet. Because your eye is fatigued to yellow, you see less of it. By comparison, you see more of the other colors present in the light reflected from the gray. These other colors add up to violet. Refer to Figure 3-1.

Do you believe it? Let's try it. Change these lines:

```
220 SETCOLOR 0,1,14
230 SETCOLOR 1,0,10
240 SETCOLOR 2,1,14
250 SETCOLOR 4,1,14
```

Run the program. Stare at the yellow area on the screen for about 30 seconds. As you continue to look at the screen, notice how the gray turns a tinge of violet. Seeing is believing.

What color would you expect the gray to turn if the background were blue?

Answer #7 ________________

Try it with a blue, orange, or green background.

#7. Orange

Contrast of Extension

Contrast of extension involves the relative areas of two or more colors. For example, a screen might be in color balance if it conveys an overall feeling of neither warm nor cold. But not all colors are equal. Smaller areas of warm colors as compared to cool colors are needed to provide this overall balance on the TV screen. For colors of the same luminance, color values could be defined as follows:

Yellow—9
Orange—8
Red—6
Violet—3
Blue—4
Green—6

Because yellow has a value of 9 and violet has a value of 3, a color balanced screen using these two colors would have three times as much violet as yellow. It is then up to the composer to convey expressions or meanings by painting screens that are out of balance. For example, if the overall effect of a screen is to convey a warm feeling, the balance would be shifted toward warm colors. This does not mean, however, that there would be a larger area of a warm color than a cool color.

Let's try an example. Change these lines:

```
220 SETCOLOR 0,8,4
230 SETCOLOR 1,8,4
240 SETCOLOR 2,2,8
250 SETCOLOR 4,2,8
```

Run the program. Although the blue and orange occupy about the same area, a warm feeling is projected.

Ok, it's time for Challenge #6.

Challenge #6

This is a double challenge. Set the colors in the color display program to:

1. Convey an overall feeling of warmth using the colors violet and red. Make the background black. (Red can have a hue of 4 and a luminance of 4; violet can have a hue of 6 and a luminance of 4)
2. Convey an overall feeling of coolness using the colors violet, and red. Make the background either violet or red.

Hint: You will need to consider the areas of the colors used.

Color GRAPHICS Modes 9, 10, 11

The graphics capabilities of the ATARI computer depend on a special video display chip (component) in the computer. Early versions of the ATARI computer used a video display chip called CTIA. Newer, updated models use a replacement of this chip called GTIA. The original chip supports BASIC GRAPHICS modes 0 through 8. The newer chip supports the color GRAPHICS modes 9, 10, and 11 as well. These additional modes provide for a larger number of colors.

The following programs will only work if you have the GTIA chip in your computer. If you don't know if you have it, give them a try to see if they work.

Mode 11

Let's begin by taking a look at the 16 colors of GRAPHICS mode 11. Here is a program to try. Don't forget to erase any old programs still in the computer.

```
100 REM 16 COLORS
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 11
220 SETCOLOR 4,0,6
300 REM DRAW 16 VERTICAL LINES
310 FOR C=0 TO 15
320 COLOR C
330 PLOT 5*C,10
340 DRAWTO 5*C,150
350 NEXT C
390 GOTO 390
400 END
```

Run it. The 16 colors (hues) that are displayed are the same as the colors listed in Table 1-1. The background is black and the luminance of the colors is set in line 220 (to the value 6 in this example). The value of the COLOR statement in line 320 ranges from 0 to 15.

You can also see that we have drawn the lines vertically. In GRAPHICS mode 11 the horizontal resolution is the same as mode 5 (80 columns) but the vertical resolution is the same as mode 8 (192 rows). Each pixel is four times as wide as it is high.

Mode 9

Mode 9 is somewhat similar to mode 11 in the way it is used. The difference is that 16 different luminances can be displayed instead of 16 different hues.

Let's change the previous program to display 16 luminances. Lines 100, 210, and 220 are changed to make your program appear as follows:

```
100 REM 16 LUMINANCES
200 REM SET UP GRAPHICS SCREEN
```

```
210 GRAPHICS 9
220 SETCOLOR 4,8,0
300 REM DRAW 16 VERTICAL LINES
310 FOR C=0 TO 15
320 COLOR C
330 PLOT 5*C,10
340 DRAWTO 5*C,150
350 NEXT C
390 GOTO 390
400 END
```

Run it. You can see that the line colors are different luminances of the background color. The background color is set in line 220 (to 8—blue) in this example.

Mode 10

We have saved mode 10 for last. With this mode, 9 different colors with 16 different luminances can be displayed. We have saved it for last because setting colors is different than with the other modes.

We have previously used the five color registers 0 to 4 with the SETCOLOR statement. But the ATARI computer actually has nine color registers. The remaining four registers cannot be assigned colors using the SETCOLOR statement. They can, however, be assigned colors using the POKE statement. POKE is a general purpose BASIC statement that is used to change the values of registers that are not normally accessible to the beginning programmer. An example of the POKE statement is POKE 704,70. This example statement instructs the computer to fill color register location 704 with color 70.

The four additional color register locations not accessible with the SETCOLOR statement are 704, 705, 706, 707. The five color registers used with the SETCOLOR statement are at locations 708 to 712. Color values can also be POKED into these locations.

The color value can range from 0 to 255. To calculate a color value, take the desired hue number from Table 1-1, multiply it by 16, and add the desired luminance value. For example, pink with a hue value of 4 and a luminance of 6 would have a color value of 70.

$(4 \times 16) + 6 = 70$

Let's look at a program example.

```
100 REM 9 COLORS AND 16 LUMINANCES
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 10
220 FOR C=704 TO 712
230 POKE C,RND(0)*256
240 NEXT C
300 REM DRAW 9 VERTICAL LINES
310 FOR C=0 TO 8
320 COLOR C
330 PLOT 5*C,10
340 DRAWTO 5*C,150
350 NEXT C
390 GOTO 390
400 END
```

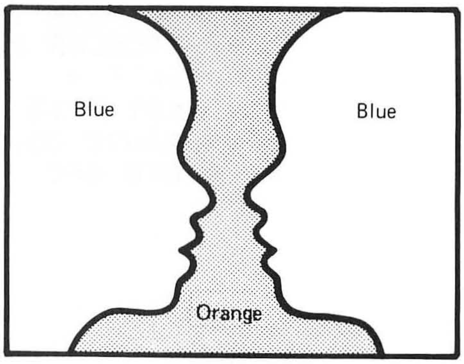
Enter the lines and run it. In this example, random colors (line 230) are POKED into the color registers. Because the colors are random, they will differ each time the program is run. Refer to a BASIC programming text for more information on the BASIC RND statement. In this mode we access the nine color registers by using values for the COLOR statement that range from 0 to 8.

Chapter 3 Review

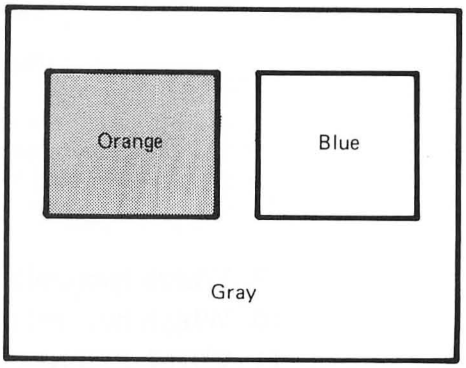
Answer the following questions.

1. Which colors provide the greatest contrast of hue: red-blue or green-blue?
2. Compared to pink, which colors are warmer: blue, yellow, violet, white, or orange?

3. Which color, blue or yellow, would best be used to convey a feeling of distance?
4. A screen has the same area of the two colors blue and orange. Is the overall balance of the screen warm or cold?
5. Which of the color GRAPHICS modes 9, 10, and 11 would most appropriately be used to provide variations of the same color?
6. What color value would you use to represent an orange hue with a luminance of 6?
7.
   - **A.** Which of the two colors would appear most distant?
   - **B.** Would this screen display most likely represent a vase or two people talking?



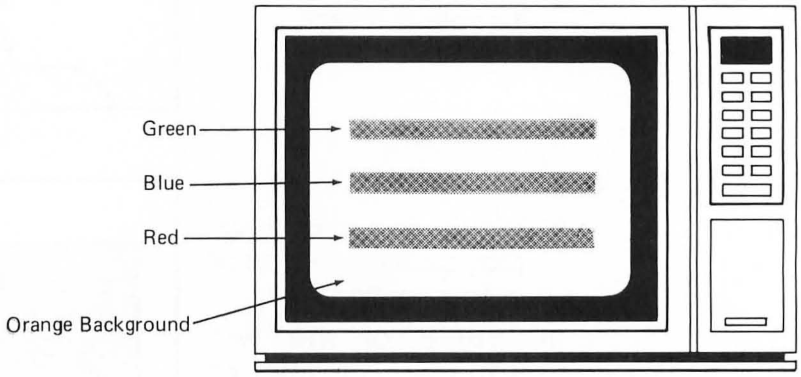
8.
   - **A.** Would this screen display project a warm or a cold feeling?
   - **B.** Which of the two squares (orange or blue) would express a feeling of being the heaviest?



The following program is used to draw three lines on the TV screen:

```
100 REM DRAW THREE LINES
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
```

```
220 SETCOLOR 4,2,8
300 REM DRAW FIRST LINE
305 SETCOLOR 0,13,12
310 COLOR 1
320 PLOT 5,5
330 DRAWTO 35,5
400 REM DRAW SECOND LINE
410 SETCOLOR 1,8,0
420 COLOR 2
430 PLOT 5,10
440 DRAWTO 35,10
500 REM DRAW THIRD LINE
510 SETCOLOR 2,4,4
520 COLOR 3
530 PLOT 5,15
540 DRAWTO 35,15
600 GOTO 600
```

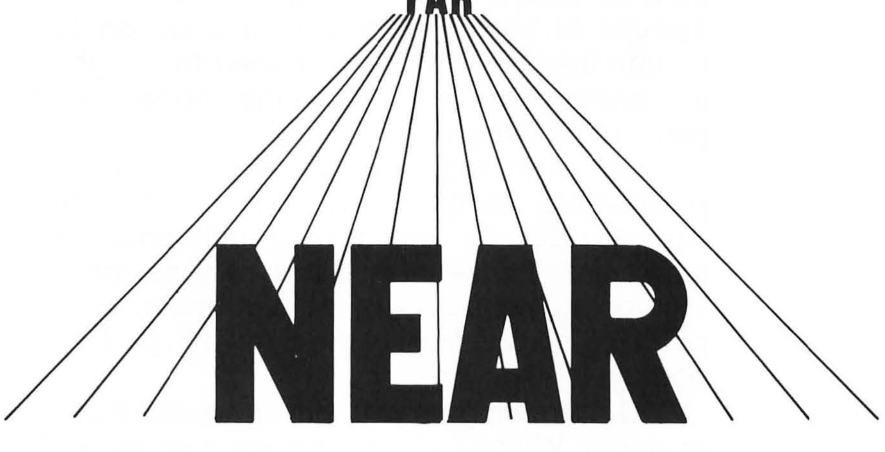


9. Which two colors provide the strongest light-dark contrast?
10. Which two colors provide the strongest contrast of hue?
11. Which of the colors is the warmest?

CHAPTER 4

Three Dimensions

Is it far or near?



What graphics techniques can be used to provide a sensation of depth?

In this chapter you will learn to:

- Draw images from a one point perspective
- Use shading to portray depth
- Draw images from a two point perspective

Television screens cannot by themselves portray depth. They are two dimensional. However, TV images can be given three dimensional characteristics. To produce a three dimensional looking image we have to fool our sense of sight. Study the drawing at the beginning of this chapter. Although the drawing is made in a two dimensional space, a feeling of depth is portrayed by the lines and different sizes of the words.

One Point Perspective

Through experience we know that near objects appear larger than far objects of the same size. Objects that are very far away appear to be no more than a point on the horizon. It's as though they have almost vanished from sight. This is called the *vanishing point.* We will use the concept of vanishing point to give our drawings depth.

Let's start with a single vanishing point or a one point perspective. Imagine a sidewalk cut into squares of the same size. A sidewalk will appear differently depending on the viewpoint. Figure 4-1 depicts two viewpoints of a sidewalk. Viewpoint A is from above, perhaps from an airplane. Viewpoint B is from in front of the sidewalk, perhaps from a standing position.

The drawing made from viewpoint B provides a sensation of depth. We will put the drawing shown in Figure 4-1(B) on the TV screen but first let's study how to construct the drawing from a one point perspective.

To start our construction, choose a vanishing point. Then draw two lines diagonally down from that point (Figure 4-2(A)). Next draw the horizontal lines representing the front and back of the first sidewalk square (Figure 4-2(B)). The remainder of the lines will be based on the drawing in Figure 4-2(B). Sketch a light construction line between the opposite corners of the first sidewalk square. Then sketch another construction line parallel to it (Figure 4-2(C)).

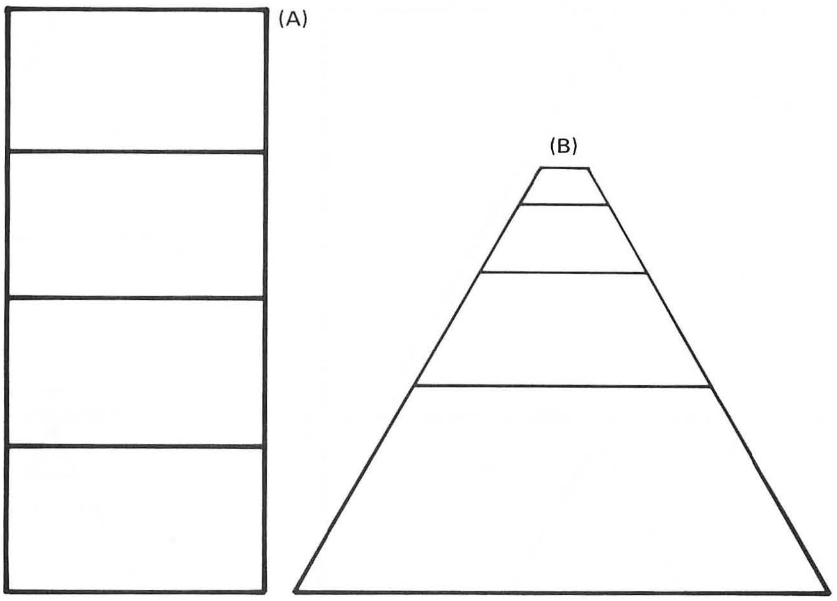
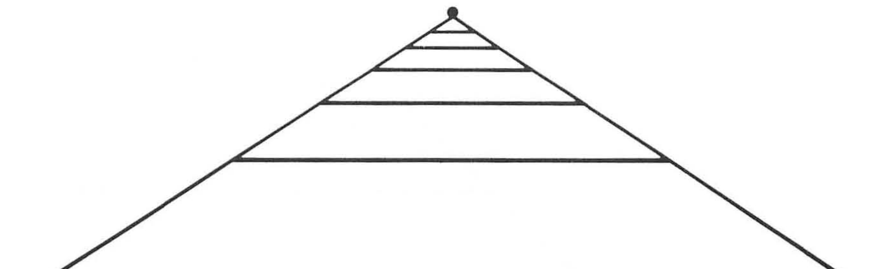


FIGURE 4-1. Two Viewpoints of a Sidewalk.

Draw a horizontal line to represent the next sidewalk square. Complete the drawing by adding as many construction lines as you need for the number of sidewalk squares you desire (Figure 4-2(D)).

You can change the appearance of the sidewalk in a couple of ways.

1. You could change the width of the sidewalk and make the viewpoint appear closer to the horizon by making the base wider.



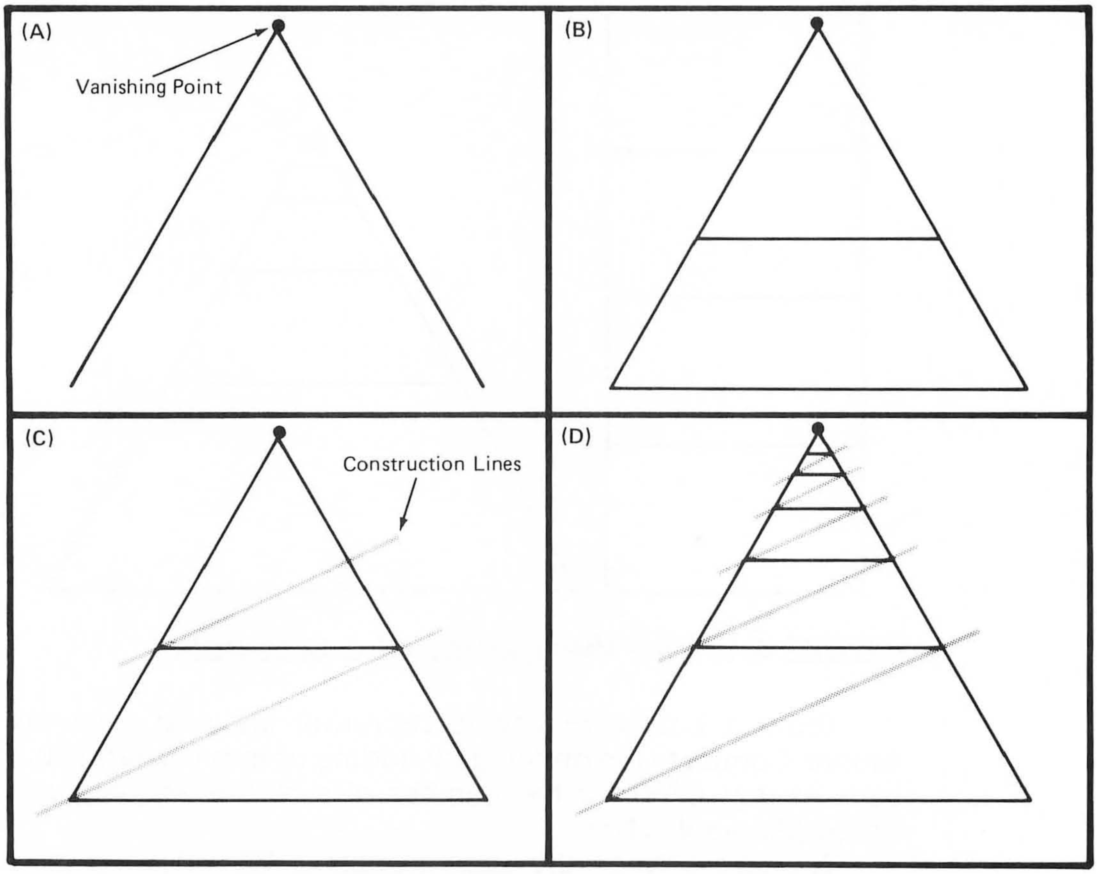
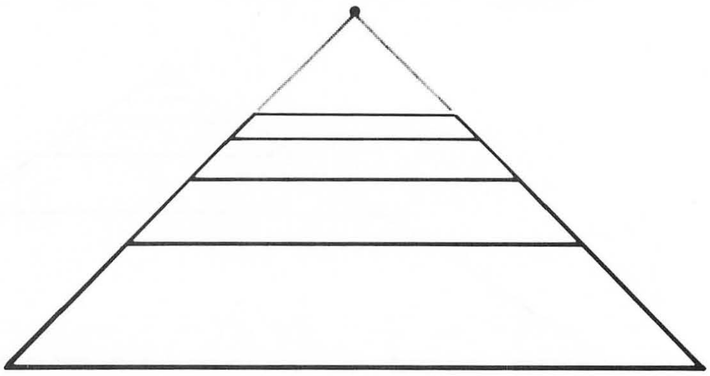


FIGURE 4-2. Construction from a one Point Perspective.

2. Your sidewalk could stop short of the vanishing point.



A Computer Sketch

Let's put the drawing of the sidewalk onto the video screen. The first step is to draw the sidewalk on a screen design worksheet. Let's do it in GRAPHICS mode 7. It might look something like Figure 4-3.

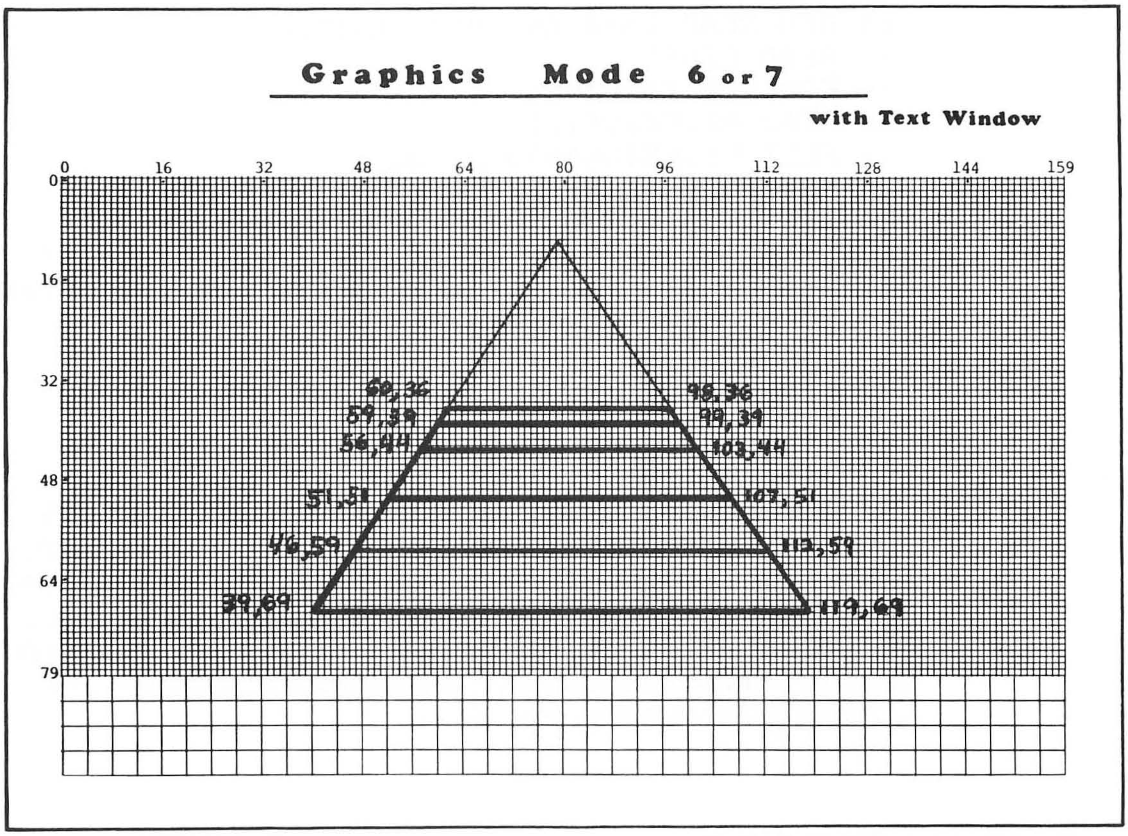


FIGURE 4-3. Layout of Sidewalk on Mode 7 Worksheet.

Notice that we are not going to draw lines all the way to the vanishing point. In fact, the vanishing point does not even have to be located on the TV screen.

Now we are ready to write a program to make our sketch. A simple way to do this is to identify the end points of each line and simply PLOT one of the end points and DRAWTO the other. Notice that the end points are labeled in the preceeding diagram. Your program might look like this:

```
100 REM SIDEWALK-1 POINT PERSPECTIVE
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 COLOR 1
300 REM READ DATA AND DRAW LINES
310 READ LINES
320 FOR L=1 TO LINES
330 READ P1,P2,D1,D2
340 PLOT P1,P2:DRAWTO D1,D2
350 NEXT L
360 DATA 8
370 DATA 60,36,98,36,59,39,99,39,56,44,103,44
380 DATA 51,51,107,51,46,59,112,59,39,69,119,69
390 DATA 60,36,39,69,98,36,119,69
800 END
```

Enter the program lines and run it.

Values of the end points are stored in DATA statements in lines 370, 380, and 390. When the READ statement is executed in line 330, four values (two graphics points) of data are read. These values are then used in line 340 to draw the line. The process is repeated for the number of lines specified in the DATA statement on line 360. If you are not familiar with READ and DATA statements, consult a BASIC programming text or the ATARI BASIC reference manual.

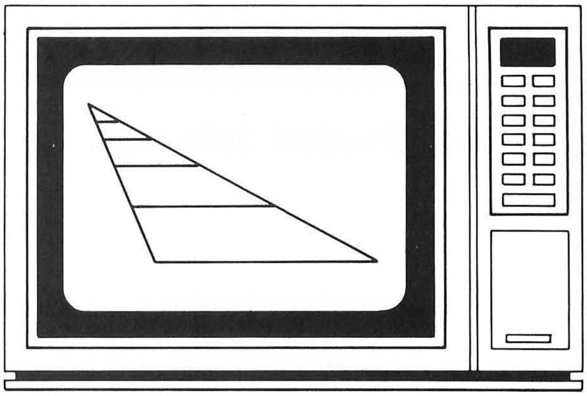
Depth by Shading

Typically, the closer an object is the better we can see it. Different luminances of the same color can be used to portray depth. The closer portion of an object would appear lighter in color than the more distant portions of the same object.

Let's use the sidewalk drawn with the previous program and shade the sidewalk squares as illustrated in Figure 4-4. We

Challenge #7

Draw a sidewalk in GRAPHICS mode 7 that is similar to the one shown here. (*Hint:* Sketch the drawing on a worksheet, identify the end points of each line, and change the values in the DATA statements of the previous program).



could draw adjacent lines to fill the desired areas with color as we did in Chapter 3. But, perhaps a simpler method in this case would be to use the color fill command. Before we shade the sidewalk, let's take a careful look at how the color fill command works. The following steps will help you to understand color fills. Try them.

1. Draw a vertical line to represent the right hand border of the fill area. It need not be straight up and down. Type these three lines.

```
GRAPHICS 7 RETURN
COLOR 1 RETURN
PLOT 90,20:DRAWTO 110,60 RETURN
```

You should see a line drawn on the screen.

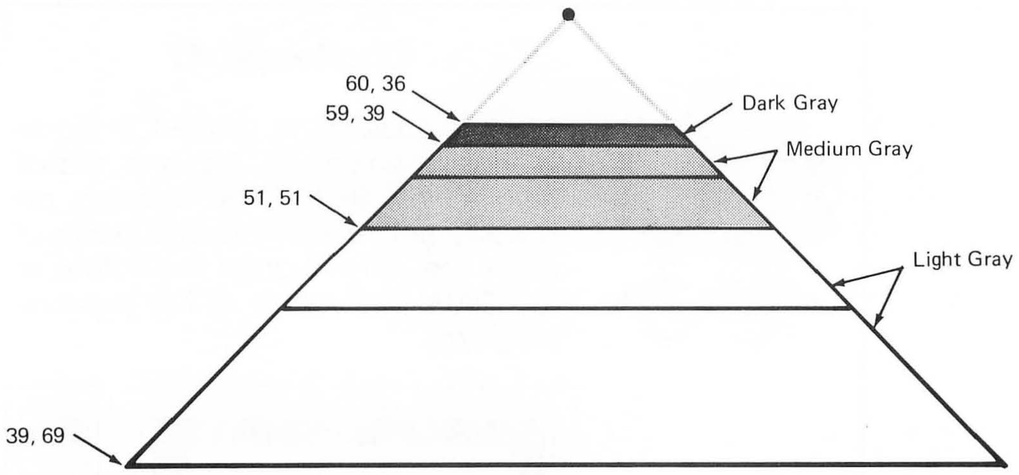


FIGURE 4-4. Shaded Sidewalk.

2. Use the POKE statement to set the fill color. Type

```
POKE 765,1 RETURN
```

The color of the fill area will be COLOR 1.

3. Use the PLOT statement to set the upper left hand corner of the fill area. Refer to Figure 4-5. Type

```
PLOT 65,40 RETURN
```

A single pixel should be plotted.

4. Use the POSITION statement to set the lower left hand corner of the fill area. Type

```
POSITION 60,50 RETURN
```

5. Execute the color fill command. Type

```
XIO 18,#6,0,0,"S:" RETURN
```

The screen should look like Figure 4-5.

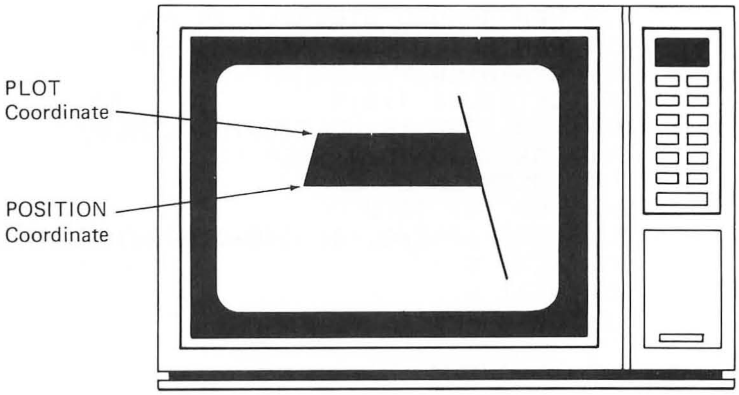


FIGURE 4-5. Color Fill Example.

Ok, let's shade the sidewalk. Add these lines to the sidewalk program:

```
400 REM SHADE SIDEWALK
500 REM SET COLORS
510 SETCOLOR 0,0,4:SETCOLOR 1,0,8:SETCOLOR 2,0,12
600 REM FILL SIDEWALK WITH 3 COLORS
630 FOR C=1 TO 3
640 READ P1,P2,F1,F2
650 POKE 765,C
660 PLOT P1,P2:POSITION F1,F2
670 XIO 18,#6,0,0,"S:"
680 NEXT C
690 DATA 60,36,59,39,59,39,51,51,51,51,39,69
```

Run it.

The points to be plotted and positioned are stored in the DATA statement at line 690. These points are read in line 640 and set on the screen in line 660.

The limitation of colors available in GRAPHICS mode 7 restricts the clarity with which the drawing can be made. If you have a GTIA chip in your computer, you can do the drawing in GRAPHICS mode 9. In mode 9 you can have up to 16 different luminances.

If you have a GTIA chip in your computer, try this example:

```
100 REM SHADED SIDEWALK IN MODE 9
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 9
220 SETCOLOR 4,0,0
300 REM DRAW 16 HORIZONTAL LINES
310 FOR C=15 TO 0 STEP -1
315 COLOR 15-C
320 FOR Y=0 TO 2
330 PLOT 69-2*C,100-3*C+Y:DRAWTO 2*C+10,100-3*C+Y
340 NEXT Y
350 NEXT C
490 GOTO 490
```

Two Point Perspective

So far our drawings have been made with reference to a single vanishing point. Another dimension can be added by drawing from a two point perspective. Let's draw a box. As we did with one point perspective, let's first see how the drawing is constructed.

Begin by choosing two vanishing points and a vertical line which will represent the front corner of the box (Figure 4-6(A)). Next draw construction lines from each vanishing point to the top and bottom of the line (Figure 4-6(B)). Now, draw two more vertical lines to mark off the front two sides of the box (Figure 4-6(C)). Complete the top of the box with two more construction lines (Figure 4-6(D)).

You know how to construct the box. We're ready to start our program. First we need to draw our box on a screen design worksheet. Let's choose GRAPHICS mode 7. Your drawing on the worksheet might look something like Figure 4-7. Notice that the end points of each line have been labeled. The program to draw the box is not much different than the one used to draw the sidewalk. In fact, the only difference is in the DATA statements.

The program looks like this. Enter the lines and try it.

```
100 REM BOX-2 POINT PERSPECTIVE
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
```

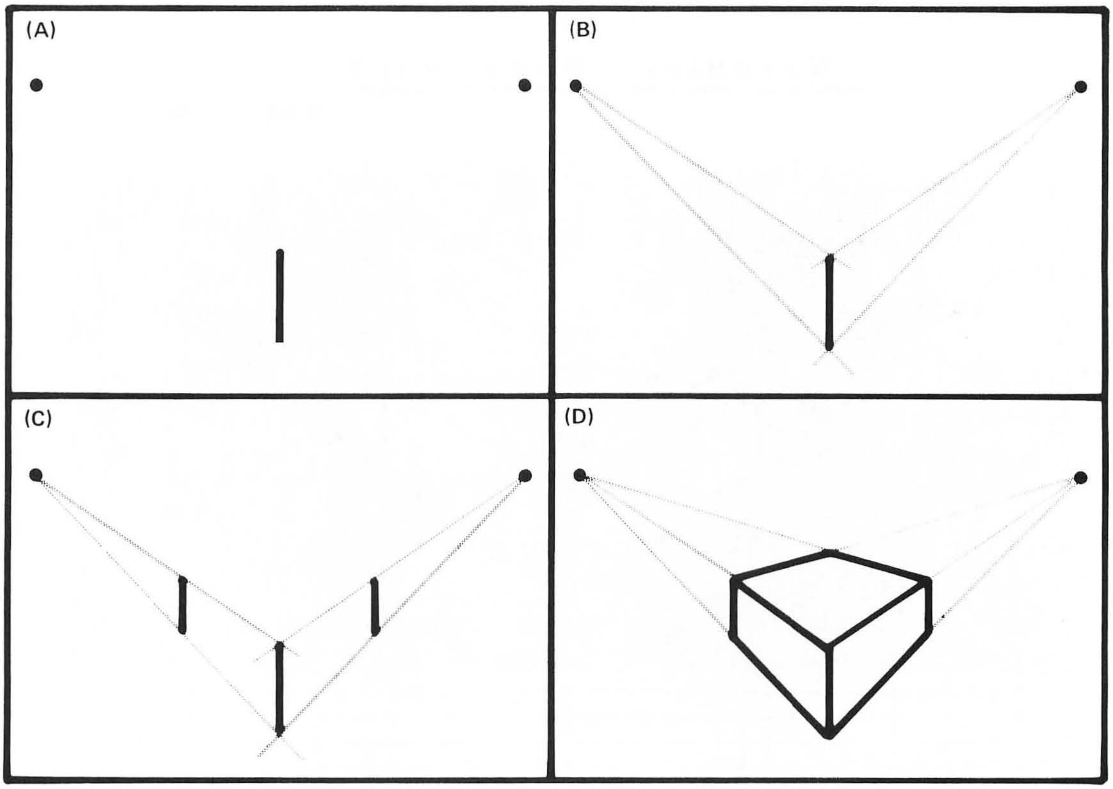


FIGURE 4-6. Construction from a Two Point Perspective.

```
220 COLOR 1
300 REM READ DATA AND DRAW LINES
310 READ LINES
320 FOR L=1 TO LINES
330 READ P1,P2,D1,D2
340 PLOT P1,P2:DRAWTO D1,D2
350 NEXT L
360 DATA 9
370 DATA 49,36,49,47,49,47,79,69,79,69,109,47
380 DATA 109,47,109,36,109,36,79,31,79,31,49,36
390 DATA 49,36,79,49,79,49,109,36,79,49,79,69
800 END
```

Let's use the color fill statement and shade one of the sides. Add these lines:

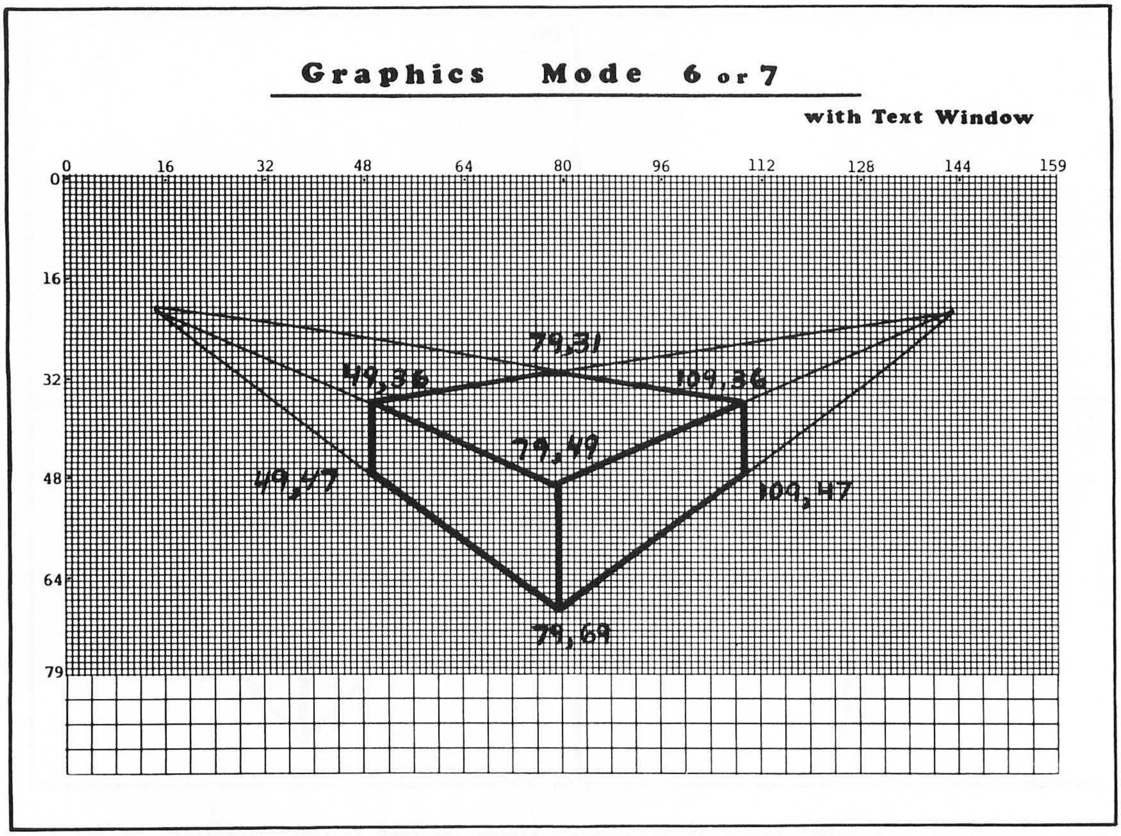


FIGURE 4-7. Layout of Box on Mode 7 Worksheet.

```
400 REM SHADE THE LEFT SIDE
500 REM SET COLORS
510 SETCOLOR 0,0,14
600 REM FILL SIDE WITH COLOR
610 POKE 765,1
620 PLOT 49,36:POSITION 49,47
630 XIO 18,#6,0,0,"S:"
640 PLOT 49,47:POSITION 79,68
650 XIO 18,#6,0,0,"S:"
```

Because the color fill only fills lines horizontally, we had to execute the color fill statement twice. Figure 4-8 shows the two areas that had to be filled separately.

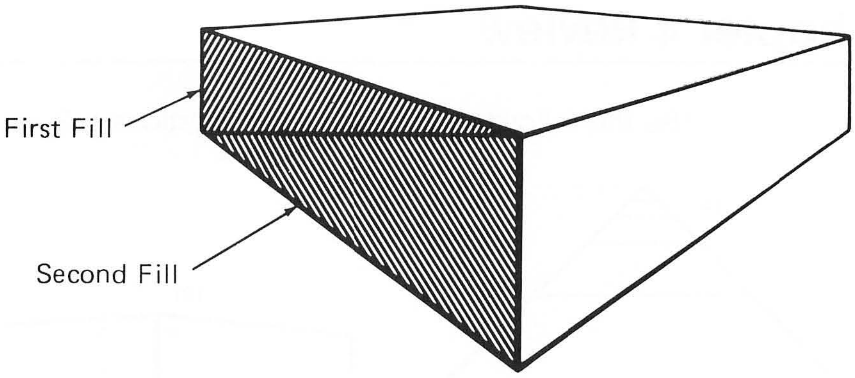
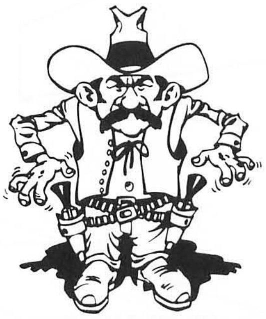
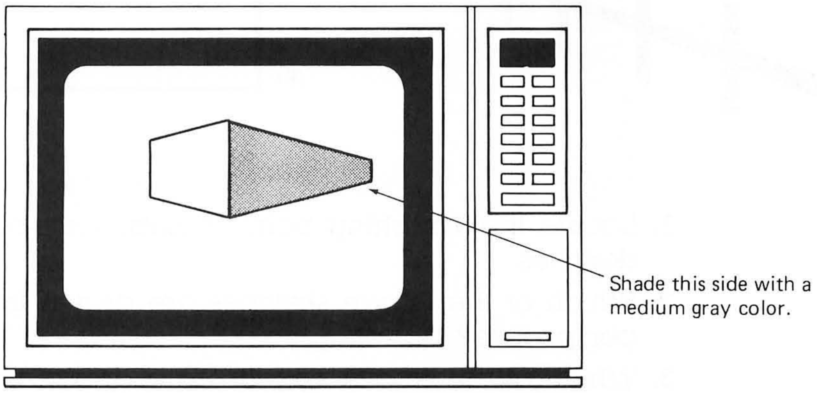


FIGURE 4-8. Color Fill for Irregular Shape.



Challenge #8

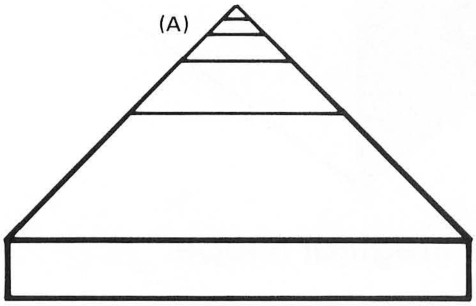
Draw the following image on the TV screen. The background should be black and the box lines should be a medium gray.



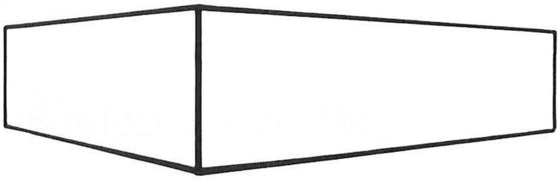
Chapter 4 Review

Use the following diagrams for questions 1, 2, and 3.

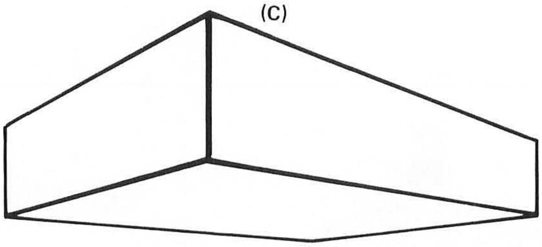
(A)



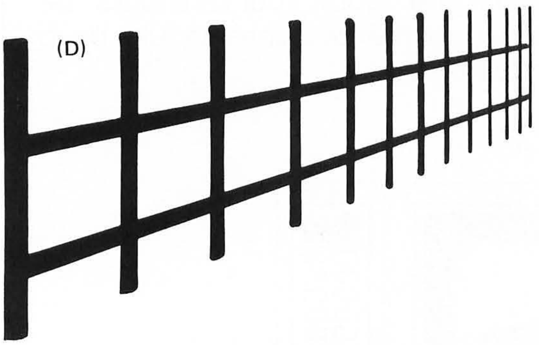
(B)



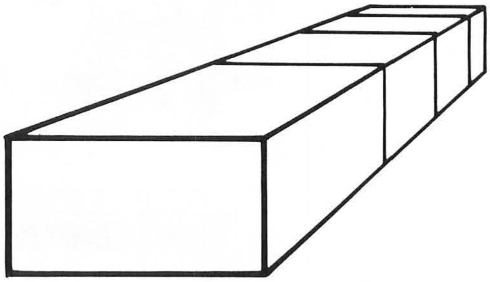
(C)



(D)



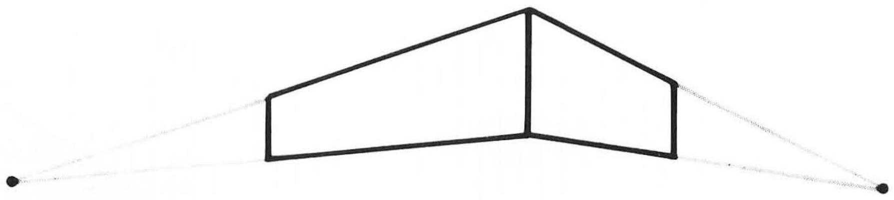
(E)



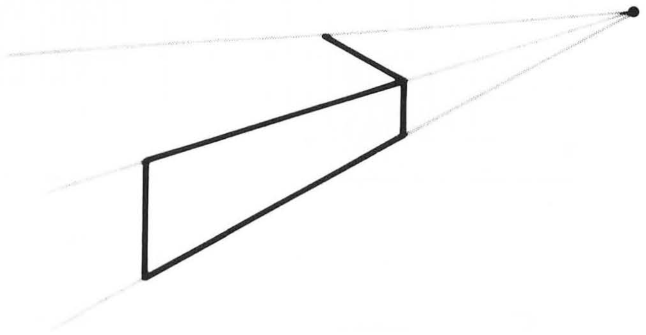
1. Locate the vanishing point (points) for each of the above sketches.
2. Which of the above sketches are drawn from a one point perspective?
3. Which of the above sketches are drawn from a two point perspective.

Complete the following constructions of boxes in questions 4, 5, and 6.

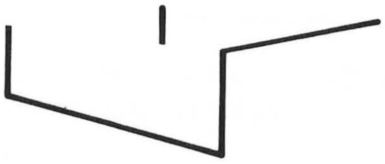
4.



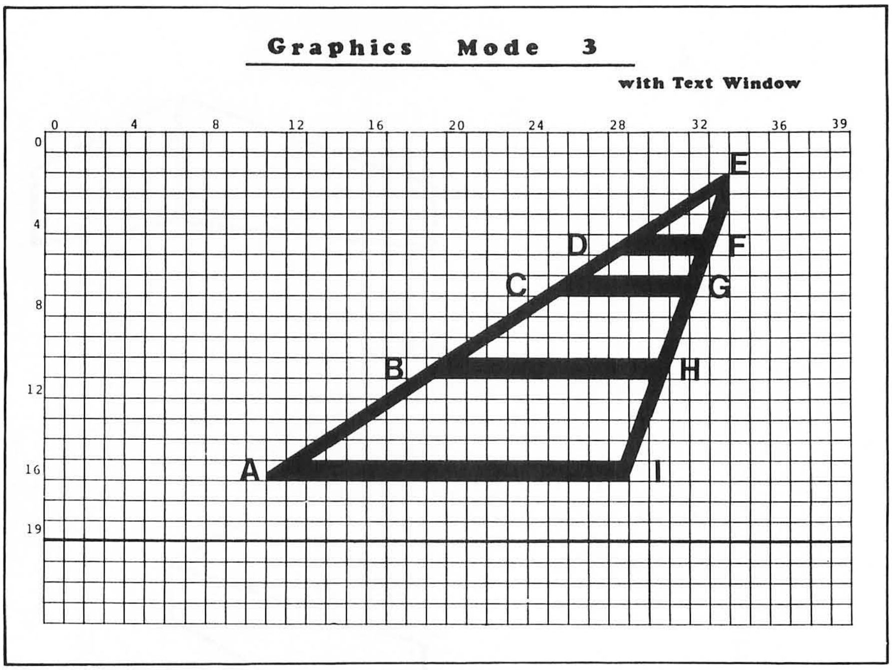
5.



6.



7. What are the endpoints for each of the lines in the following sketch? The sketch is drawn on a mode 3 worksheet.



A. ______________ **B.** ______________

C. ______________ **D.** ______________

E. ______________ **F.** ______________

G. ______________ **H.** ______________

I. ______________

8. Sketch the screen image that will be produced by the following color fill program.

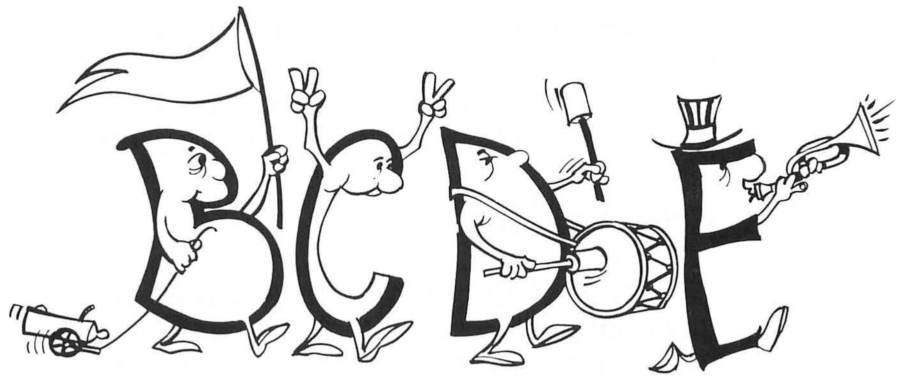
```
100 REM COLOR FILL PROBLEM
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
300 REM DRAW RIGHT BORDER
```

```
310 COLOR 1
320 PLOT 100,50:DRAWTO 140,10
330 POKE 765,1
340 PLOT 70,10:POSITION 30,50
350 XIO 18,#6,0,0,"S:"
```

CHAPTER 5

Characters

Let's take a look at some characters.



Sometimes information cannot be depicted pictorially as it was in Chapters 2, 3, and 4. The clever use of characters and text will often add clarity to the screen and provide information that cannot be easily expressed with pictures.

In this chapter you will learn to:

- Use GRAPHICS modes 0, 1, and 2
- Change the color of characters
- Produce graphics designs made of characters

ATARI Characters

What is a character? Characters are generally thought to be either letters or numbers. But they can also be punctuation marks or graphics designs. Later in this book we will design our own characters, but for now we will use the characters stored in the computer. Table 5-1 shows the set of characters that the ATARI computer normally uses. Each character has a decimal code associated with it. These codes will be of use later.

The three text modes supported by ATARI BASIC are GRAPHICS modes 0,1, and 2. A GRAPHICS mode 0 text window is also supported in MODES 1 through 8. For each of these GRAPHICS modes, the characters shown in Table 5-1 can normally be printed on the screen.

Each of the characters shown on the keyboard can be easily used by simply depressing the desired key. Special keys or combination of keys are required to obtain lower case and graphics characters. A diagram of the keyboard is shown in Figure 5-1.

The ATARI keyboard is explained in detail in the ATARI owner's manual. A brief description is given here.

Graphics Characters

Let's try some graphics characters. Note the position of the **CTRL** key near the left side of the keyboard. Hold the **CTRL** key down while pressing keys for several of the letters A through Z. The symbols you see on the screen are called *graphics characters.*

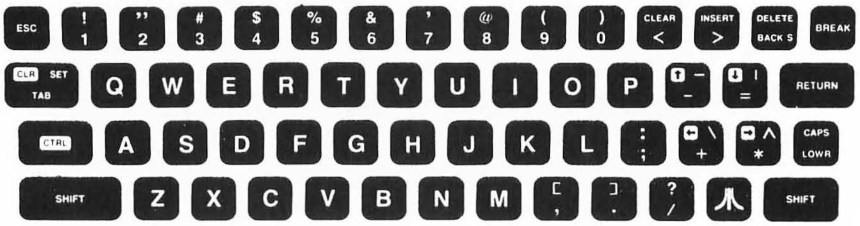


FIGURE 5-1. ATARI Keyboard. From the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

TABLE 5-1. ATARI Character Set

Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character	Decimal Code	Character
0	♥	13	▔	26	└	39	'	52	4	65	A	78	N	91	[	104	h	117	u
1	├	14	▁	27	␛	40	(	53	5	66	B	79	O	92	\	105	i	118	v
2	▕	15	▖	28	↑	41	)	54	6	67	C	80	P	93	]	106	j	119	w
3	┘	16	♣	29	↓	42	*	55	7	68	D	81	Q	94	∧	107	k	120	x
4	┤	17	┌	30	←	43	+	56	8	69	E	82	R	95	_	108	l	121	y
5	┐	18	─	31	→	44	,	57	9	70	F	83	S	96	♦	109	m	122	z
6	╱	19	┼	32	Space	45	–	58	:	71	G	84	T	97	a	110	n	123	♠
7	╲	20	●	33	!	46	.	59	;	72	H	85	U	98	b	111	o	124	|
8	◢	21	▄	34	"	47	/	60	<	73	I	86	V	99	c	112	p	125	↰
9	▗	22	▎	35	#	48	0	61	=	74	J	87	W	100	d	113	q	126	◀
10	◣	23	┬	36	$	49	1	62	>	75	K	88	X	101	e	114	r	127	▶
11	▝	24	┴	37	%	50	2	63	?	76	L	89	Y	102	f	115	s		
12	▘	25	▌	38	&	51	3	64	@	77	M	90	Z	103	g	116	t		

Adapted from the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

Lower Case

To get lower case letters, press the key labeled **CAPS LOWR**. You do not need to hold it down as with the **CTRL** key. Press several of the characters A through Z.

To return the keyboard to upper case characters, hold the **SHIFT** key while pressing the **CAPS LOWR** key. Pressing **SYSTEM RESET** will also return the keyboard to upper case.

Inverse Video

Each of the characters used by the ATARI computer can be displayed in inverse video. This is a reversal between background and character luminance. Figure 5-2 shows the letter H in both normal and inverse video.

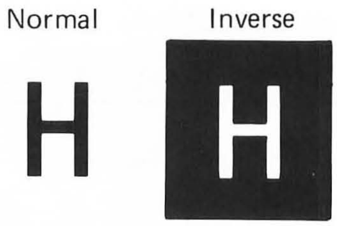


FIGURE 5-2. Inverse Video Character.

To get inverse video characters, press the key. Try it and then press the keys for several of the characters A through Z. To return the keyboard to the normal state, press the key again.

GRAPHICS Mode 0

Let's try a program to print some characters on the screen. Enter the following lines. Don't forget the semicolon in line 30.

```
10 GRAPHICS 0
20 FOR X=1 TO 760
30 PRINT "A";
40 NEXT X
```

Run it. You should see 20 rows of A's. Each row contains 38 characters. Twenty rows of 38 characters per row equals 760 characters. Notice that line 20 of our program executes the FOR . . . TO, NEXT loop 760 times. Also notice that the two character positions on the left are not used. They are normally not used because these two positions do not show up well on some TVs. In a moment we will show how to use these two positions but first let's return to our program and print a different character than the letter A.

Change line 30 to

```
30 PRINT "+";
```

Hold **CTRL** and press S to get this character.

Run it. It should look like a piece of graph paper. You could try other graphics characters to get a variety of designs.

Characters can also be printed without actually depressing the key representing the desired character. Each character in the computer is internally represented by a number. A standard exists for this representation. It is called the ASCII code (American Standard Code for Information Interchange). The decimal code (ASCII value) for each ATARI character was shown in Table 5-1. Since graphics characters have no standard, the codes in this table are called the ATARI ASCII code (ATASCII).

Let's try an example. The ATASCII code for the letter A is 65 (refer to Table 5-1). To use the ATASCII code from ATARI BASIC, we can use the CHR$ function. Type the following line.

```
PRINT CHR$(65) RETURN
```

Was the letter A printed? It should have been. The CHR$ function lets us represent any character by a decimal value.

?

What ATASCII code would be used to print the letter Z? Refer to Table 5-1.

Answer #1 ______________

Let's modify the previous program to print a screen full of A's using the CHR$ function. Your program should look like this:

```
10 GRAPHICS 0
20 FOR X=1 TO 760
30 PRINT CHR$(65);
40 NEXT X
```

Try it. You should see 20 lines of A's just like before.

The characters in Table 5-1 can be printed in inverse video by adding 128 to the decimal code of the character. For example, an inverse video A would have a decimal code of 65 plus 128 or 193. Let's try it. Change line 30 to

```
30 PRINT CHR$(193);
```

Run it. Is the screen filled with inverse video A's?

Let's print several different characters. Make these changes to the program:

```
20 FOR X=65 TO 90

30 PRINT CHR$(X);
```

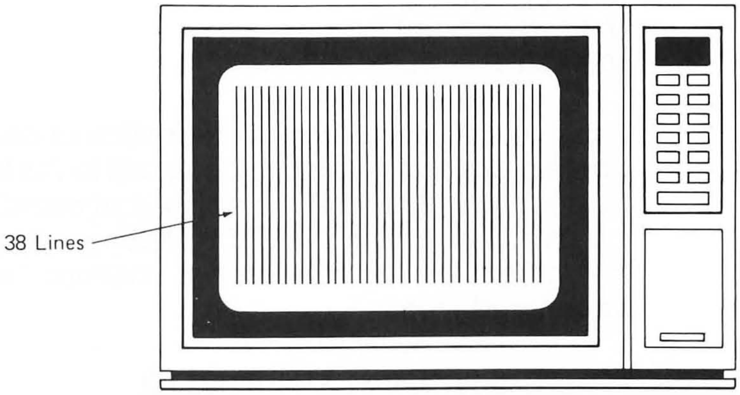
Run it. You should see the upper case alphabet printed at the top of the screen. Do you see the power of representing characters by numbers, or in this case, a variable?

Ok, Challenge #9.

#1. 90

Challenge #9

Write a program to produce the following screen display using mode 0 characters.



Character Position

The position of characters on the screen can be controlled in several ways. Let's try a couple of examples. Clear the screen. Hold the **SHIFT** key down and press the **CLEAR** key. Now type

```
PRINT "TEST";"WORDS" RETURN
```

Were the two words printed next to each other without a space between them? They should have been. Now try this statement:

```
PRINT "TEST","WORDS" RETURN
```

The use of the comma has caused the second word to be shifted to the right. Actually, the comma causes the printing to be shifted to the next print zone. The print zone positions are normally set 10 columns apart.

The comma and the semicolon only control horizontal spacing. We can use the POSITION statement to control both horizontal and vertical placement of characters. Let's try an example. Clear the computer of any old programs and enter this one:

```
10 GRAPHICS 0
20 FOR P=20 TO 10 STEP -1
30 POSITION P,P
40 PRINT "TEST"
50 NEXT P
```

Run it. We started the printing position at coordinate 20,20. The printing position was moved one unit to the left and one unit up for each print line until we stopped at coordinate 10,10.

We can use the POSITION statement to print anywhere on the screen, even in the first two columns. Let's try it. Clear the screen and type

```
POSITION 0,5:PRINT "TEST" RETURN
```

We can also change the left hand margin to use the two columns at the left of the screen. Type

```
POKE 82,0 RETURN
```

The left margin for all printing is now set at column 0. The number 82 in the POKE statement is a place where the computer keeps track of the column number for the left margin. If we change it to 0 with the POKE statement, our text will start in column 0 on the screen. We could change it to any column number that we like. We can also change the right margin. Try it. Type

```
POKE 83,6 RETURN
```

Now press a key and hold it down until a bunch of characters are printed. Are the margins set at 0 and 6? You can get things back to normal by pressing **SYSTEM RESET**.

What statement would you use to change the right margin to column 3?

Answer #2 ______________

GRAPHICS Modes 1 and 2

As you might recall from Chapter 1, GRAPHICS modes 1 and 2 are used for printing colored characters with sizes larger than GRAPHICS mode 0 characters. Let's print some GRAPHICS mode 2 characters on the screen. Enter and run this program:

```
10 GRAPHICS 2
20 FOR X=1 TO 200
30 PRINT#6;"*";
40 NEXT X
```

How do the number of columns and rows compare to those of GRAPHICS mode 0? Since each character is twice the width and twice the height, there are only half as many rows and half as many columns. Try GRAPHICS mode 1. Change line 10 to

```
10 GRAPHICS 1
```

Run the program again. How do the sizes of mode 1 and mode 2 characters compare?

So far, all of the characters we've printed in modes 1 and 2 have had an orange color. Both of these GRAPHICS modes are five color modes. This means that five colors can be displayed on the screen. The background is one of the five colors. This leaves four colors for the characters.

The following program will print the word "COLOR" in four different colors. Enter the lines and run it.

#2. POKE 83,3

```
10 GRAPHICS 2
20 PRINT #6;"COLOR"  ←  upper case
30 PRINT #6;"COLOR"  ←  inverse upper case
40 PRINT #6;"color"  ←  lower case
50 PRINT #6;"color"  ←  inverse lower case
```

Notice that only upper case characters were printed. We'll print lower case characters in a moment.

The colors of the characters are set by color registers 0 through 3. Table 5-2 shows the various character types and their associated color registers.

TABLE 5-2. Character Color Registers

Character Type	SETCOLOR Register #	Default Color
Upper case alphabetical	0	Orange
Lower case alphabetical	1	Light Green
Inverse upper case alphabetical	2	Dark Blue
Inverse lower case alphabetical	3	Red
Numbers	0	Orange
Inverse numbers	2	Dark Blue

Adapted from the ATARI® 400/800 BASIC REFERENCE MANUAL by permission of Atari, Inc. © 1980.

The colors of the characters can be changed by using the SETCOLOR statement to change the values of the color registers of Table 5-2. Let's try it. Type

```
SETCOLOR 0,8,2 RETURN
```

Color register 0 holds the color value for the upper case alphabet colors. The characters that were orange are now dark blue.

What statement would you use to set lower case alphabet characters to dark blue?

Answer #3 ________________

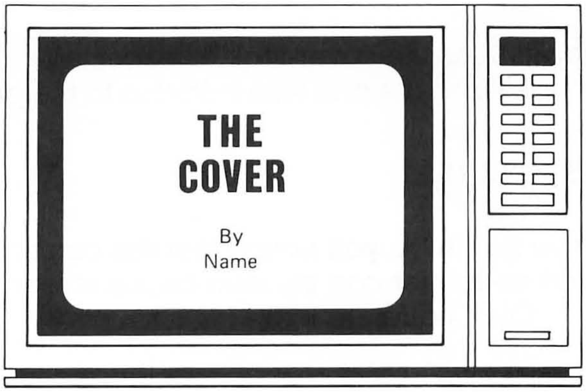
So far, only upper case characters have appeared on the screen. Upper and lower case characters cannot normally be printed on the screen simultaneously. We can, however, use the POKE statement to change all printed letters to lower case. Try it. Type

```
POKE 756,226 RETURN
```

The letters on the screen should have changed to lower case letters. You've probably noticed all of those hearts. They have filled the space where no characters were written. For now, we can only get rid of them by printing another character over them. In Part Two we'll see how to change them to empty spaces by modifying the character set.

Cover Page

Let's try a small project. Suppose we want to make a nice cover for a program we're writing. It might look something like this:



#3. SETCOLOR 1,8,2 (answer may vary)

We will use a split screen with GRAPHICS mode 2. Let's make the background and text window black. The words "THE" and "COVER" will be light blue. Print your name in the text window. Let's answer some questions to see if you understand.

What GRAPHICS statement will you use with this program?

Answer #4 ________________

What SETCOLOR statement will you use to turn the text window black?

Answer #5 ________________

The program might look something like this:

```
10 REM COVER PAGE
20 GRAPHICS 2
30 SETCOLOR 2,0,0:SETCOLOR 0,9,10
40 POSITION 8,2:PRINT #6;"THE"
50 POSITION 7,4:PRINT #6;"COVER"
70 PRINT "                   By"
80 PRINT "                Thomas"
100 GOTO 100
```

Try it. Does the cover look as expected?

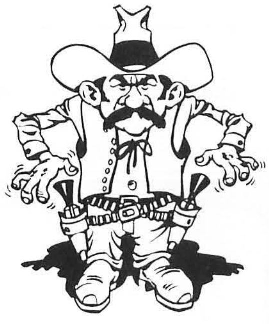
Let's make one little addition to the program. Add this line:

65 POKE 752,1

Now run it. Do you notice that the cursor has become invisible and no longer can be seen on the screen.

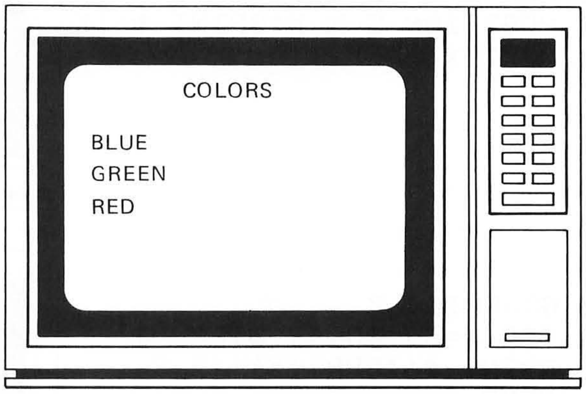
Ok, Challenge #10.

#4. GRAPHICS 2 #5. SETCOLOR 2,0,0



Challenge #10

Write a program to produce the following screen display in full screen GRAPHICS mode 2. The B is blue, the G is green, and the R is red. All of the other letters should be orange. The background should be black.



Chapter 5 Review

Match the following screen displays with the appropriate program.

1. ________

```
10 GRAPHICS 2
20 FOR X=5 TO 1 STEP -1
30 POSITION 5-X,X:PRINT #6;X
40 NEXT X
```

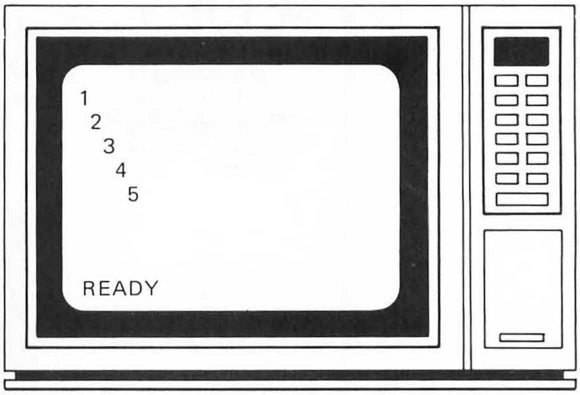
2. ________

```
10 GRAPHICS 2
20 FOR X=54 TO 49 STEP -1
30 PRINT #6;CHR$(X)
40 NEXT X
```

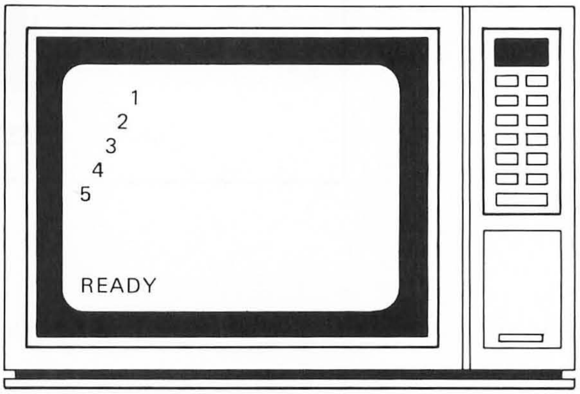
3. ________

```
10 GRAPHICS 2
20 FOR X=1 TO 5
30 POSITION X,X:PRINT #6;X
40 NEXT X
```

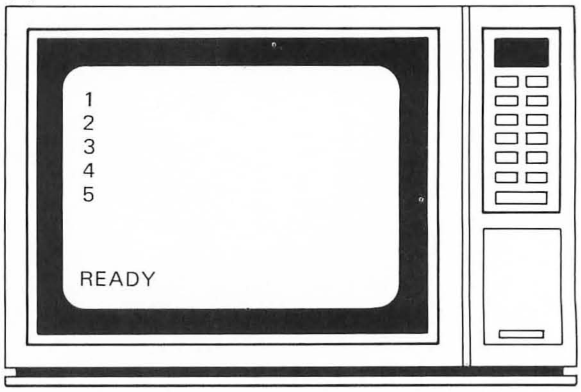
(A)



(B)



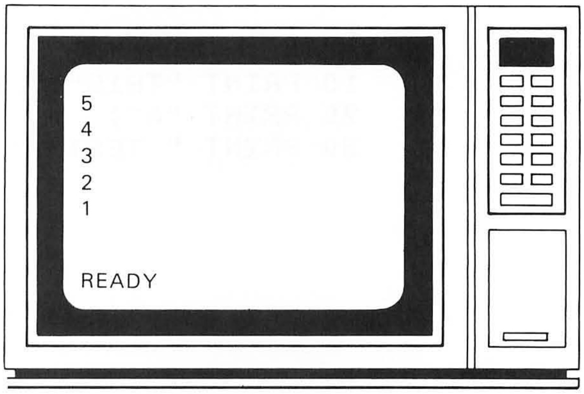
(C)



4. ________

```
10 GRAPHICS 2
20 FOR X=49 TO 54
30 PRINT #6;CHR$(X)
40 NEXT X
```

(D)



5. What characters would be printed by the following statements?

PRINT CHR$(72) ________

PRINT CHR$(6) ________

PRINT CHR$(198) ________

If the following characters are printed in GRAPHICS mode 1, what color would they normally appear and what SETCOLOR register would be used to change their color?

		Default Color	SETCOLOR Register	
6.	C	a) ________	b) ________	(Inverse Video)
	o	a) ________	b) ________	
	L	a) ________	b) ________	
	o	a) ________	b) ________	(Inverse Video)
	r	a) ________	b) ________	

7. What statement is used to set the right margin to 22 in GRAPHICS mode 0?

8. What statement is used to turn the cursor off?

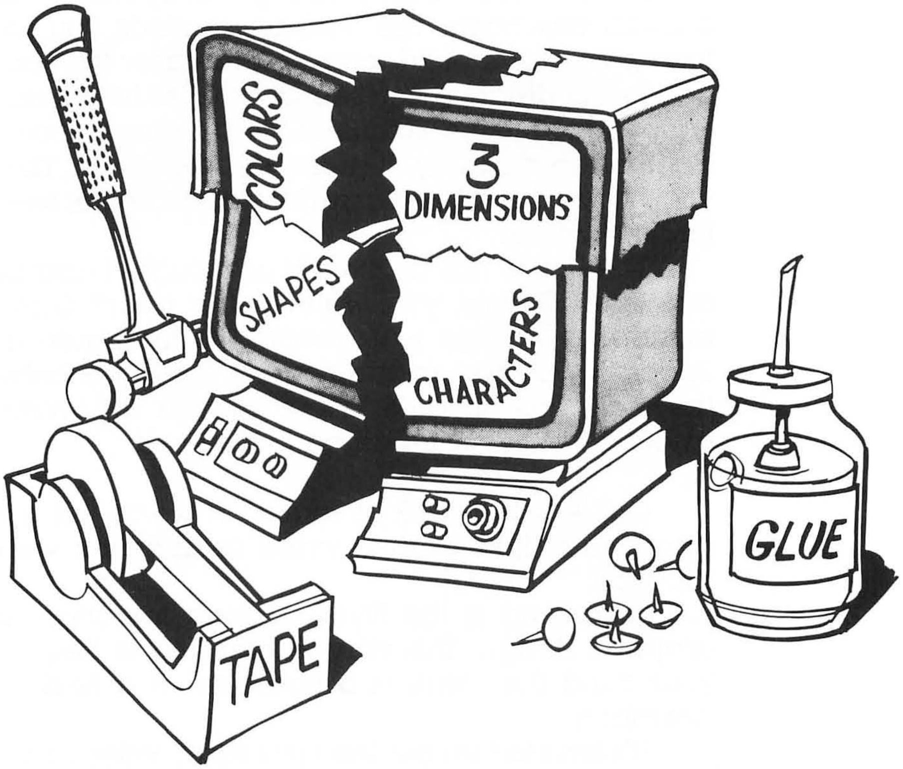
9. What screen image will the following program produce?

```
10 PRINT "THIS";" IS"
20 PRINT "A";
30 PRINT " TEST"
```

CHAPTER 6

A Video Composition

It's time to put the pieces together.



In this chapter you will learn to:

- Translate a story into a picture
- Project various visual sensations
- Design a video artpiece

In the first five chapters we studied ATARI graphics design and programming using the BASIC language. At this point, you should have a feeling for using color and characters as well as designing three dimensional shapes. Our primary goal in this chapter will be to combine all of the ideas presented in Chapters 1 through 5 into a single video composition. Let's begin with a story.

You feel relaxed as you sit in the large padded chair. As your mind wanders, you are suddenly far away from home. You find yourself walking through an open field. It is late on an autumn afternoon. The cool air reminds you that summer is long past. The ground seems barren after the fall harvest.

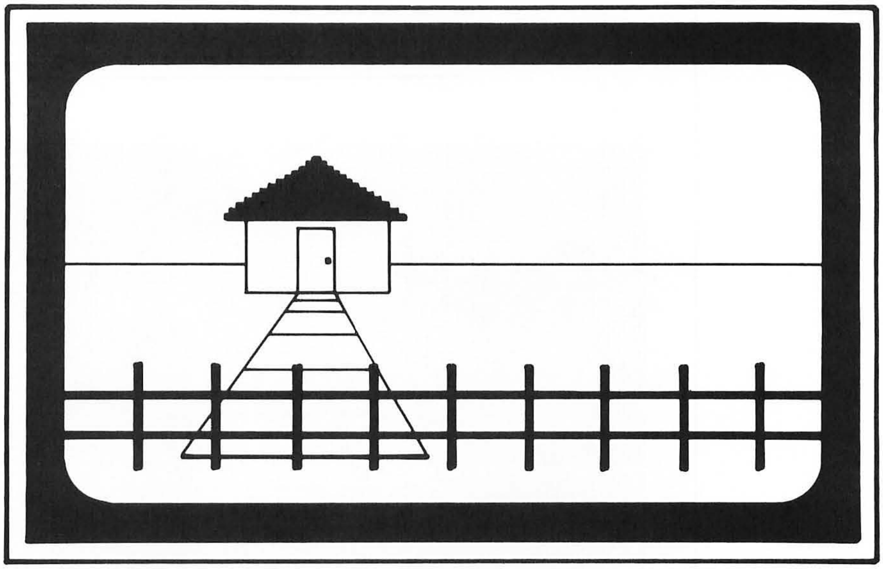
In the distance you see what looks like a building. As you walk toward it, the outline of the building becomes magnified against the clear sky. It seems to be a house. The front is plain with no windows. The large door at the center seems to provide the only access.

The house has no life. It's as though it had been deserted for years. Should you walk to the door? Something seems mysterious. A wide walk leading to the house is divided into impossibly perfect rectangles. But, the fence between you and the walk has no gate. It's as though an invitation is being extended but you are dared to accept it.

Ok, let's get back to reality. This was just a story, but it is also a description of the video composition we are about to produce.

Imagining is the first and most important part of video graphics design. This story is intended to help you picture in your mind the scene of a house set in a field on an autumn afternoon.

It's time to turn our thoughts into a video composition. Here is a visual interpretation of the short story describing the house.



It may not be the same interpretation as you would have drawn, but let's use the drawing to aid us in writing our computer program.

We need to assign colors to the various parts of the drawing. Let's assume that our computer does not have a GTIA chip and that only four colors are available to us at one time.

The sky is clear and feels far away. Let's choose a cool color like light blue for it. Since it's autumn and late afternoon, let's make the ground a light brown or orange color. We have used two colors and so we have two colors remaining. We could color the house bottom and fence one color (white, perhaps) and color the house roof and sidewalk with the remaining color (gray, perhaps). This is enough information to begin our program. We will return later to change the colors and observe the effect.

Program Modules and Organization

Before beginning the program, let's sketch our drawing on a screen design worksheet. What GRAPHICS mode will we use?

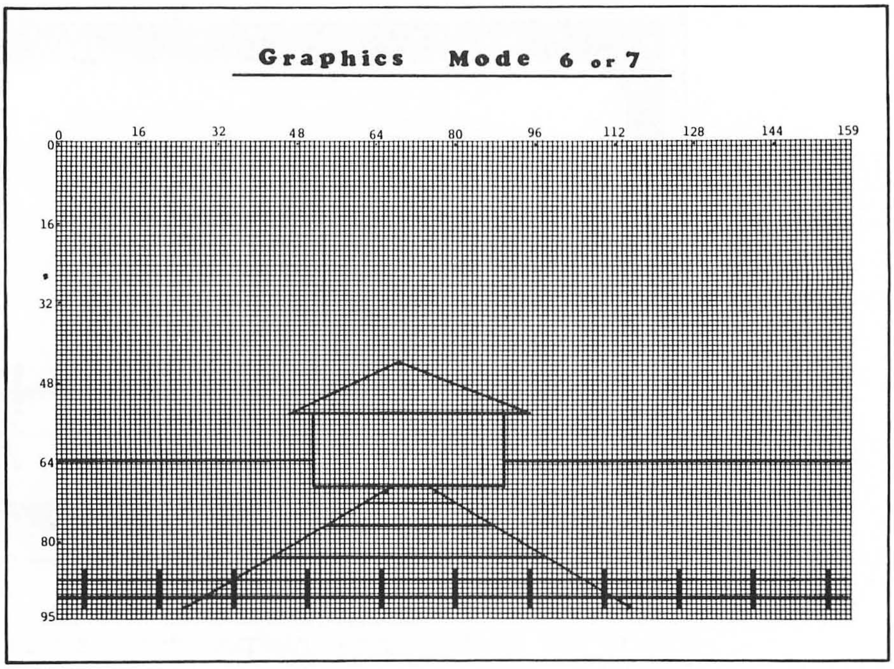


FIGURE 6-1. Layout of House Scene on Mode 7 Worksheet.

Since we will be using four colors, we need to select a four color mode such as GRAPHICS mode 3, 5, or 7. Since mode 7 has the highest resolution, let's make it our choice. The sketch on a mode 7 worksheet might look like Figure 6-1.

Let's write the program piece by piece, all the time remembering such things as color, size, and balance. To make the entire composition easier to understand, we will design the program in modules. For example, the routine to draw the house might be one module, the background another module, and so on. These modules will be as independent of each other as possible. They will give a certain degree of structure to our program.

Before beginning our composition, let's lay out the structure of the entire program. We can use the format of a program to express it.

Lines	Remarks
100–190	Title and comments
200–290	Set GRAPHICS mode, initialize colors
1000–1900	Draw background
2000–2900	Draw house
3000–3900	Draw sidewalk
4000–4900	Draw fence

Each of the program modules will be written into the block of corresponding line numbers.

Title

Let's document the program with a couple of REM statements. Enter these lines.

```
100 REM THE HOUSE - A VIDEO COMPOSITION
110 REM WRITTEN ON DATE BY NAME
```

Set Mode and Color

Color registers 0, 1, 2, and 4 are used in GRAPHICS mode 7. We can represent the sky with register 4 which is the background color. It makes little difference which colors are assigned to the other color registers. Add these lines to your program:

```
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7+16
220 SETCOLOR 0,1,10
230 SETCOLOR 1,0,14
240 SETCOLOR 2,0,6
250 SETCOLOR 4,9,10
```

Color register 0 will be used for the ground, register 1 will be used for the house bottom and fence, register 2 will be used for the house top and sidewalk, and register 4 will be used for the sky.

The Background

We are ready to begin drawing. Our first task will be to draw the background. The color of the sky has been set in line 250. The ground color is set by color register 0 so the lines we draw for the ground must utilize the COLOR 1 statement. We can make the block of color by drawing adjacent horizontal lines as we did in Chapter 3. The module might look as shown below. Enter the lines:

```
1000 REM BACKGROUND MODULE
1010 COLOR 1
1020 FOR Y=65 TO 95
1030 PLOT 0,Y:DRAWTO 159,Y
1040 NEXT Y
```

Add this line to the end of the program so that we can run it without the display returning to GRAPHICS mode 0.

```
9000 GOTO 9000
```

Run the program. Do you have a blue sky and an orange ground?

The House

We are ready to draw the house. First let's do the roof. We can use the color fill statement as we did in Chapter 4. Enter these lines:

```
2000 REM HOUSE MODULE
2100 REM HOUSE ROOF
2110 COLOR 3
2115 POKE 765,3
2120 PLOT 95,55:DRAWTO 70,45
2130 POSITION 45,55
2140 XIO 18,#6,0,0,"S:"
```

We can draw the bottom of the house by using the same technique as we used to draw the ground. Enter these lines:

```
2200 REM HOUSE BOTTOM
2210 COLOR 2
2220 FOR Y=56 TO 70
2230 PLOT 50,Y:DRAWTO 90,Y
2240 NEXT Y
```

Run the program. Does the house have a gray roof and a white bottom? Is your composition beginning to take shape?

The Sidewalk

We know all about drawing sidewalks. We can put the endpoints of each sidewalk line into DATA statements just as we did in Chapter 4. The sidewalk module looks like this.

```
3000 REM SIDEWALK MODULE
3010 COLOR 3
3020 READ LINES
3030 FOR L=1 TO LINES
3040 READ P1,P2,D1,D2
3050 PLOT P1,P2:DRAWTO D1,D2
3060 NEXT L
3090 DATA 5
3100 DATA 65,71,25,93,75,71,115,93,61,73,79,73
3110 DATA 53,78,86,78,41,85,99,85
```

Add the lines to your program and run it. Does it look as you expected?

The Fence

The fence consists of two horizontal lines and a series of vertical lines. Let's draw the two horizontal lines first and then draw the series of vertical lines by using a loop. Here is the module.

```
4000 REM FENCE MODULE
4010 COLOR 2
4015 REM HORIZONTAL FENCE LINES
4020 PLOT 0,89:DRAWTO 159,89
4030 PLOT 0,91:DRAWTO 159,91
4035 REM VERTICAL FENCE LINES
4040 FOR X=5 TO 155 STEP 15
4050 PLOT X,87:DRAWTO X,93
4060 NEXT X
```

Add the lines to your program. Run it. Does the screen look like the design from the mode 7 worksheet?

A Little Extra

The Door

The drawing is complete but it seems to be missing something. Oh, we've forgotten the door. Let's go back to the house module and add these lines:

```
2300 REM HOUSE DOOR
2310 COLOR 3
2320 PLOT 65,70:DRAWTO 65,60:DRAWTO 75,60:
DRAWTO 75,70
```

Run it. Does it look better?

A Bird

The sky seems so empty. Let's put a bird in the sky. Add these lines:

```
5000 REM BIRDS
5010 COLOR 2
5020 PLOT 110,30:DRAWTO 114,27:DRAWTO 118,31
5025 DRAWTO 122,27:DRAWTO 126,30
```

Try the program again. Does it look more balanced?

A Frame

It looks good enough to frame. Let's put a border around it. Add these lines:

```
6000 REM A BORDER
6005 COLOR 3
6010 FOR X=0 TO 1
6020 PLOT X,X:DRAWTO 159-X,X:DRAWTO 159-X,95-X
6025 DRAWTO X,95-X:DRAWTO X,X
6030 NEXT X
```

Run it. Does everything look good? If so, congratulations. You have just completed your video artpiece. Here is a list of the entire program:

```
100 REM THE HOUSE - A VIDEO COMPOSITION
110 REM WRITTEN ON DATE BY NAME
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7+16
220 SETCOLOR 0,1,10
230 SETCOLOR 1,0,14
240 SETCOLOR 2,0,6
250 SETCOLOR 4,9,10
1000 REM BACKGROUND MODULE
1010 COLOR 1
1020 FOR Y=65 TO 95
1030 PLOT 0,Y:DRAWTO 159,Y
1040 NEXT Y
2000 REM HOUSE MODULE
2100 REM HOUSE ROOF
2110 COLOR 3
2115 POKE 765,3
2120 PLOT 95,55:DRAWTO 70,45
2130 POSITION 45,55
2140 XIO 18,#6,0,0,"S:"
2200 REM HOUSE BOTTOM
2210 COLOR 2
2220 FOR Y=56 TO 70
2230 PLOT 50,Y:DRAWTO 90,Y
2240 NEXT Y
2300 REM HOUSE DOOR
2310 COLOR 3
2320 PLOT 65,70:DRAWTO 65,60:DRAWTO 75,60:DRAWTO 75,70
```

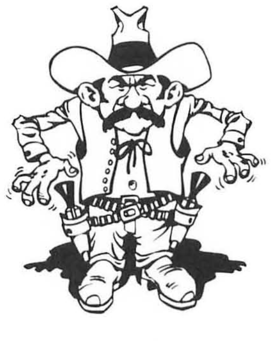
```
3000 REM SIDEWALK MODULE
3010 COLOR 3
3020 READ LINES
3030 FOR L=1 TO LINES
3040 READ P1,P2,D1,D2
3050 PLOT P1,P2:DRAWTO D1,D2
3060 NEXT L
3090 DATA 5
3100 DATA 65,71,25,93,75,71,115,93,61,73,79,73
3110 DATA 53,78,86,78,41,85,99,85
4000 REM FENCE MODULE
4010 COLOR 2
4015 REM HORIZONTAL FENCE LINES
4020 PLOT 0,89:DRAWTO 159,89
4030 PLOT 0,91:DRAWTO 159,91
4035 REM VERTICAL FENCE LINES
4040 FOR X=5 TO 155 STEP 15
4050 PLOT X,87:DRAWTO X,93
4060 NEXT X
5000 REM BIRDS
5010 COLOR 2
5020 PLOT 110,30:DRAWTO 114,27:DRAWTO 118,31
5025 DRAWTO 122,27:DRAWTO 126,30
6000 REM A BORDER
6005 COLOR 3
6010 FOR X=0 TO 1
6020 PLOT X,X:DRAWTO 159-X,X:DRAWTO 159-X,95-X
6025 DRAWTO X,95-X:DRAWTO X,X
6030 NEXT X
9000 GOTO 9000
```

Experiment

At this point, you might want to experiment with some of the colors. We will try one example and leave the rest of the experimentation to you. Change these lines:

```
220 SETCOLOR 0,0,10
250 SETCOLOR 4,0,0
```

Notice that all of the colors are neutral. An interesting effect, don't you think?



Challenge #11

Write a routine to add a cloud to the upper left part of the sky in your video composition. Line 7000 would be a good place to put the module.

A Summary

Chapters 1 through 6 have been a simple introduction to designing graphics screens and writing graphics programs in the BASIC language. They have been intended for beginners with perhaps a little programming experience. Many exciting graphics displays can be produced with relatively simple BASIC programs.

The ability to write programs is only part of producing good video graphics. The ability to understand color, shape, balance, and feeling also plays a major role.

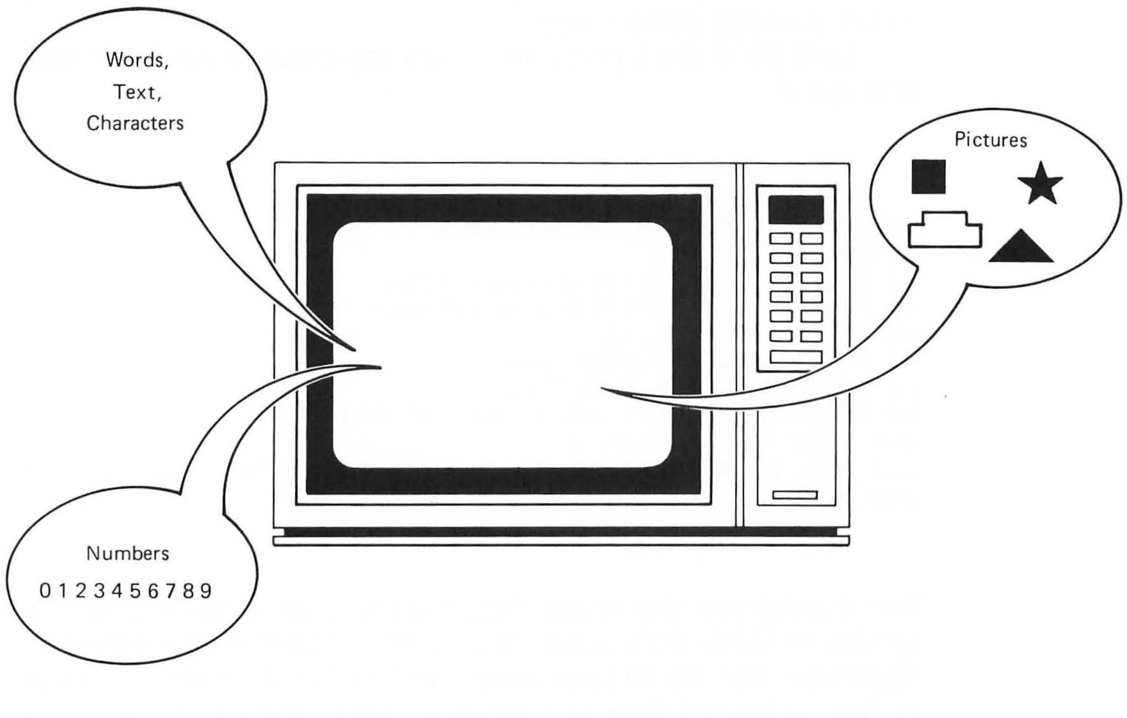
Although good graphics displays can be produced with elementary BASIC programming, the most exciting graphics facilities of the ATARI computer are only available through somewhat sophisticated programming. In Part Two you will learn to unleash the most exciting graphics hardware of the ATARI computer. The ideas, however, are conceptually more difficult to understand than those in Part One. If you do not have a reasonably good programming background, you may find it difficult. But, you be the judge. If you're unsure as to whether you can handle it, give it a try and best of luck.

PART TWO

CHAPTER 7

Pictures and Words— Mixing Modes

Get ready to combine pictures with words by mixing GRAPHICS modes.



In this chapter you will learn to:

- Explain the television display system
- Design mixed mode displays
- Build a display list
- Use mixed mode display techniques

What is this mixing modes stuff? Here is an answer. Mixing modes is displaying more than one GRAPHICS mode on a single screen. In Chapters 1 through 6, we used a number of different GRAPHICS modes. Only one GRAPHICS mode was used at a time. An exception to this was when split screen modes were used. In that case, a GRAPHICS mode 0 text window was used along with a particular GRAPHICS mode.

But, suppose that we are interested in placing GRAPHICS mode 2 text on the upper portion of the screen and perhaps drawing something in GRAPHICS mode 7 on the bottom of the same screen. This would require a mixing of GRAPHICS modes 2 and 7 on the same screen.

Let's try a short program as an example. Enter each line and run it.

```
10 REM MIX MODE DEMONSTRATION
20 GRAPHICS 7+16
30 DLIST=PEEK(560)+PEEK(561)*256
40 POKE DLIST+3,71:POKE DLIST+6,7
60 POKE DLIST+92,65
70 POKE DLIST+93,PEEK(560)
80 POKE DLIST+94,PEEK(561)
90 POKE 87,2:POSITION 8,0:PRINT #6;"TEST"
100 POKE 87,7:COLOR 2
110 FOR Y=10 TO 60 STEP 3:PLOT 5,Y:DRAWTO 155,Y:NEXT Y
200 GOTO 200
```

You should see the word "TEST" displayed at the top of the screen in GRAPHICS mode 2. The remainder of the screen is filled with horizontal lines drawn in GRAPHICS mode 7. A look at the technical side will provide some insight into what is happening.

TV Display Generation

A brief look into the workings of your TV and your ATARI computer is essential for a good understanding of this chapter. To begin, we must understand the method by which televisions display images on the TV screen. The following is a simplified explanation.

Inside of your television picture tube are several parts. At the back of the tube is an electron gun. This gun is used to shoot a stream of electrons at the front of the picture tube. We will refer to the front of the picture tube as the screen. The screen contains a phosphor coating. This coating emits light when struck by the stream of electrons. Figure 7-1 shows a simplified view of a typical TV picture tube.

Since the electron beam has a very small diameter, a stream of electrons shot at the phosphor coating would normally only light a small dot on the screen. But magnetic deflection coils attached to the tube deflect the electron beam so that it can scan the entire screen.

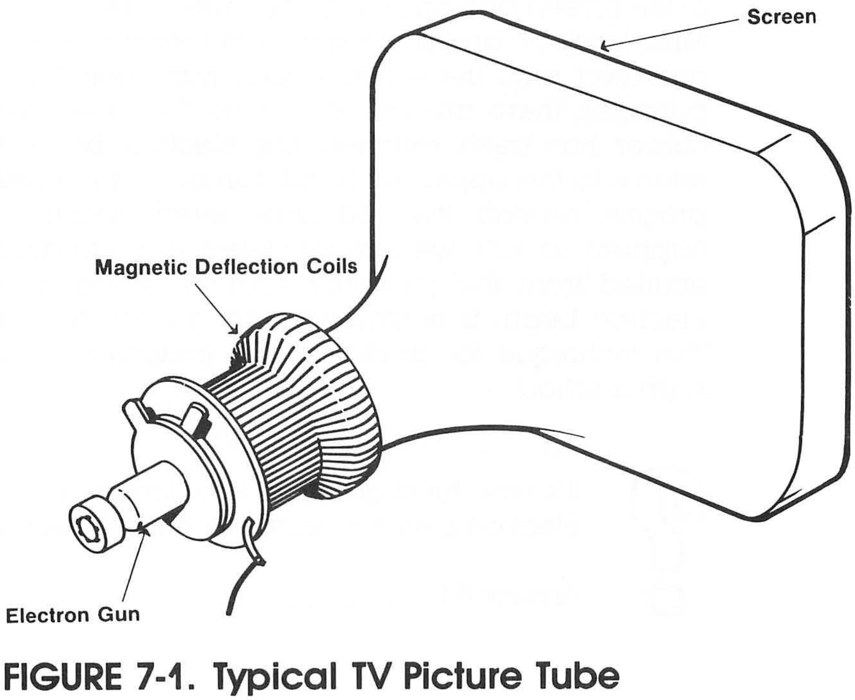


FIGURE 7-1. Typical TV Picture Tube

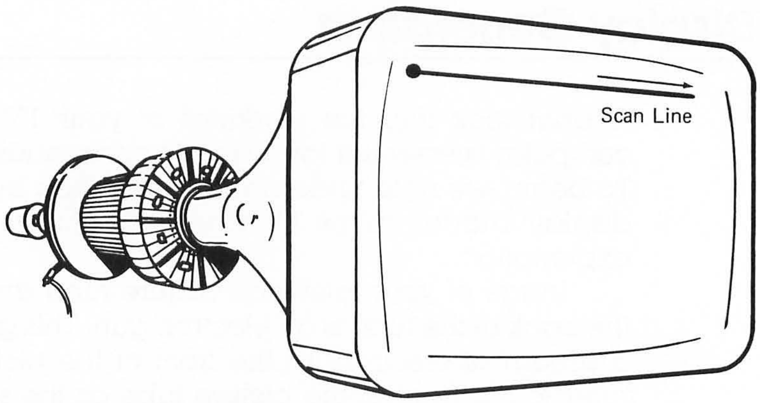


FIGURE 7-2. Scan Line

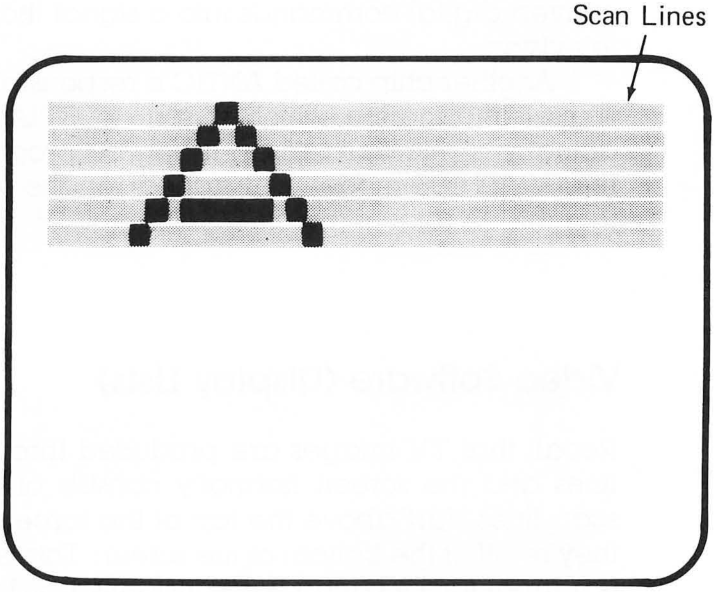
The electron beam scans in a very systematic way. First, it starts at the upper left hand corner of the screen and is rapidly deflected across the screen to the upper right hand corner. This horizontal deflection causes a scan line to be produced (see Figure 7-2). The beam then turns off and returns to the left side of the screen and one scan line lower. The electron beam then turns back on and scans across the second line. This process is repeated until the entire screen has been scanned. For our purposes, there are 262 scan lines. After the lower right hand corner has been reached, the electron beam turns off and returns to the upper left hand corner of the screen. This entire process repeats itself 60 times every second. Because this happens so fast, we cannot detect any variation in the light emitted from the phosphor coating. It seems as though the electron beam is continually hitting each spot on the screen. This technique for displaying TV images is called the *raster scan method.*

? It's time for a question. How long would it take for the electron beam to scan the entire screen once?

Answer #1 ________________

#1. 1/60 second

Scanning alone will not produce images on the screen. Images are produced by varying the intensity of the electron beam as it sweeps across the screen. For example, the letter A would be formed a scan line at a time by turning the electron beam on at specific places in the scan line.



Complex images with varying luminance levels can be formed in this way. Note that the process is somewhat more complex than described here, especially with colored displays.

ATARI Video Generation

Information displayed on a TV screen must be generated in some way. Let's take a look at the ATARI video generator and how it produces images on the TV screen.

Video Hardware

Several hardware components in the ATARI computer are either directly or indirectly responsible for displaying a computer generated image on the TV screen. One of the components responsible for video generation is the CTIA or GTIA chip studied in Part One. The primary function of this chip is to convert digital commands into a signal that can be sent to the television.

Another chip called ANTIC is responsible for producing the digital information which is sent to CTIA or GTIA. ANTIC is actually a microprocessor. It can be programmed to change the GRAPHICS modes that appear on the screen. It is this chip that this chapter is about.

Video Software (Display Lists)

Recall that TV images are produced through the use of scan lines and the screen normally consists of 262 scan lines. The scan lines start above the top of the screen and continue until they're off of the bottom of the screen. This is called *overscan*. It is normal for TVs and keeps unsightly borders from appearing. When using TVs with computers it is important that information is not lost at the top or bottom of the screen. Therefore, ANTIC normally uses only 192 of the 262 scan lines when displaying information and, thus, a border appears. However, more may be used as we'll see later. The box used in GRAPHICS 0 uses 192 scan lines. The borders above and below the box account for the remainder of the 262 scan lines.

Several scan lines may be required to form a pixel or a character on the screen. For example, a GRAPHICS mode 7 pixel requires two scan lines while a GRAPHICS mode 3 pixel requires eight scan lines. A group of scan lines is called a *mode line*. So, a GRAPHICS 3 mode line is made up of eight scan lines.

As mentioned earlier, ANTIC can be programmed. The program used with ANTIC is called a *display list*. The display list is a set of instructions that determines the GRAPHICS modes to be displayed on the screen. The display list is stored in RAM and can be changed.

The ATARI BASIC language is designed to automatically build a display list. This is accomplished by executing a graphics statement such as GRAPHICS 0. In order to mix GRAPHICS modes, however, we need to write a display list program of our own or customize an existing BASIC display list.

Before we attempt this program, let's take a look at a display list that has been written by the computer with the BASIC statement GRAPHICS 0. Enter and run the following program:

```
10 REM LOOK AT GRAPHICS 0 DISPLAY LIST
15 GRAPHICS 0
20 DLIST=PEEK(560)+PEEK(561)*256
30 FOR M=DLIST TO DLIST+31
40 PRINT PEEK(M)
50 NEXT M
```

I know, it printed so fast you lost part of it off the top of your screen. Because the list is easier to understand if it's listed vertically, let's use the one shown below rather than printing it horizontally on the screen. It should look something like this. Comments have been added for use later.

```
112 }
112 }→ Blank eight lines (three times)
112 }
66  ——→ LMS Instruction
64  }→ Address of screen display RAM
156 }  (These values will be different if you don't have 48K of
2   |  RAM in your computer)
2   |
2   |
2   |
2   |→ Mode bytes for GRAPHICS 0
2   |
2   |
2   |
2   |
2   |
```

$$\left.\begin{array}{l}2\\2\\2\\2\\2\\2\\2\\2\\2\\2\\2\\2\\2\end{array}\right\}\longrightarrow \text{Mode bytes for GRAPHICS 0}$$

$$65 \longrightarrow \text{Jump instruction}$$

$$\left.\begin{array}{l}32\\156\end{array}\right\}\longrightarrow \begin{array}{l}\text{Address of display list RAM}\\ \text{(These values will be different in you don't have 48K}\\ \text{of RAM in your computer)}\end{array}$$

Let's take a close look at what this all means.

Each of the values listed represents a machine language instruction or a memory address. ANTIC generates the screen format by using these values to set up the appropriate mode lines. Note that the values shown are decimal representations. Also note that since ANTIC is a microprocessor, it has its own instruction set.

Each of the first three instructions tell ANTIC to blank eight scan lines. They are called *blank instructions.* This takes care of the overscan on the TV and brings our computer picture down to a readable position.

The next instruction is called the LMS (Load Memory Scan) instruction. It tells ANTIC to load the following two values (bytes). The following two values will be the address at which the screen display RAM begins. The LMS instruction also tells ANTIC which mode will be used for the first mode line. Other options available with this instruction will be discussed later.

After the values representing the screen RAM location come the values corresponding to each of the mode lines which will be displayed on the screen. In this case, the value 2 is used to represent GRAPHICS 0 mode lines. Since there are 24 mode lines in GRAPHICS mode 0, 23 of these values are required in addition to the LMS instruction which specified the first mode line.

Finally, the last instruction puts our display list program into an infinite loop. It is called the JMP (Jump) instruction. It tells ANTIC to jump to the beginning of the display list which starts at the address represented by the next two values (bytes).

This last step finishes the display list program. Since an entire screen is scanned 60 times per second, this display list program is executed 60 times per second. ANTIC will continue to use the instructions in this program until the values are changed. The values can be changed by executing a BASIC graphics instruction or by changing individual portions of the display list either through BASIC or by machine language routines.

Building a Custom Display List

Before building a custom display list, you must first decide on what the screen is to look like. The display list worksheet in Appendix A should be used as an aid.

As an example, let's design a screen with two lines of GRAPHICS mode 2 at the top of the screen followed by four lines of GRAPHICS mode 0 (see Figure 7-3). The remainder of the screen will be GRAPHICS mode 3. We will use this example throughout the entire procedure for building a display list.

The information in Table 7-1 will be required as the procedure for building the display list is used. The table will not be explained here but will be used as a reference for the remainder of the chapter.

We will build the display list using the BASIC language.

1. Our first task is to coordinate the desired number of mode lines with the required number of scan lines. Refer to Table 7-1. These have already been laid out in our example. The important point is that the mode lines should normally add up to produce 192 scan lines. The following table shows the mode line and scan line requirements for our example.

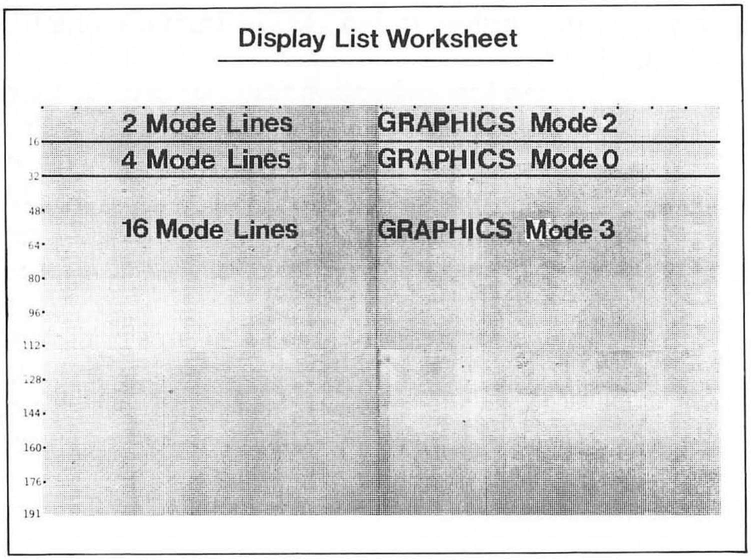


FIGURE 7-3. Layout of Mode Lines.

TABLE 7-1. ANTIC Mode Line Requirements*

ANTIC Mode	BASIC Mode	No. of Colors	Scan Lines/ Mode Line	Pixels/ Mode Line	Bytes/ Line	Bytes/ Screen
2	0	2	8	40	40	960
3	none	2	10	40	40	760
4	none	4	8	40	40	960
5	none	4	16	40	40	480
6	1	5	8	20	20	480
7	2	5	16	20	20	240
8	3	4	8	40	10	240
9	4	2	4	80	10	480
10	5	4	4	80	20	960
11	6	2	2	160	20	1920
12	none	2	1	160	20	3840
13	7	4	2	160	40	3840
14	none	4	1	160	40	7680
15	8	2	1	320	40	7680

*Reprinted from the De Re Atari Guide by permission of Atari, Inc.

BASIC Mode	No. of Mode Lines		No. of Scan Lines per Mode Line		Total No. of Scan Lines
2	2	×	16	=	32
0	4	×	8	=	32
3	16	×	8	=	128
					192

Let's see if you understand the relationship between mode line and scan line.

How many scan lines would be required for three GRAPHICS 5 mode lines?

Answer #2 ________________

2. The next step is to determine the ANTIC mode byte for each of the desired BASIC modes. Refer to Table 7-1. Be careful not to confuse the ANTIC mode with the BASIC mode.

BASIC mode	ANTIC mode byte
2	7
0	2
3	8

What is the ANTIC mode byte for BASIC mode 7?

Answer #3 ________________

3. Set the LMS option. This option is selected by setting bit 6 of a mode instruction byte. Since bit 6 translates to a decimal

#2. 12 scan lines #3. 13

value of 64, we take the ANTIC mode byte value for the mode line at the top of the screen and add 64 to it.

For our first BASIC mode 2 line the byte that includes the LMS instruction will be

LMS option(64) + ANTIC mode byte (7) = 71

?

What would be the value of the LMS byte if the first mode line on the screen was GRAPHICS 7?

Answer #4 ________________

Let's take a look at what our memory map of the display list should look like.

Code	*Remarks*	
112	Blank 8 lines	
112	Blank 8 lines	
112	Blank 8 lines	
71	Set LMS option and display ANTIC mode 7 (BASIC mode 2)	
?	PEEK(88)	memory address
?	PEEK(89)	of screen RAM
07	ANTIC mode 7 (BASIC mode 2)	
02		
02		
02	4 lines of BASIC mode 0	
02		
08		
08		
08		
08		
08		
08	16 lines of BASIC mode 3	
08		
08		
08		
08		

#4. 77

```
08
08
08
08
08
08
65  Jump to beginning of display list
 ?  PEEK(560) | memory address of          |
 ?  PEEK(561) | beginning of display list  |
```

Notice that we were unable to fill in the memory address of the screen RAM and the display list. The operating system, however, keeps track of these addresses for us.

The decimal memory addresses 88 and 89 represent the low and high bytes, respectively, of the screen RAM address. The decimal memory locations 560 and 561 represent the low and high bytes, respectively, of the display list address.

4. At this point you could simply use the POKE statement and a BASIC program to set up your custom display list. The routine might look something like this:

```
10 REM MIX MODE EXAMPLE
15 GRAPHICS 0
20 DLIST=PEEK(560)+PEEK(561)*256
30 REM 3*8 BLANK LINES
40 POKE DLIST,112
50 POKE DLIST+1,112
60 POKE DLIST+2,112
70 REM SET LMS OPTION
80 POKE DLIST+3,71
90 REM SCREEN MEMORY LOCATION
100 POKE DLIST+4,PEEK(88)
110 POKE DLIST+5,PEEK(89)
120 REM ANOTHER MODE 2 LINE
130 POKE DLIST+6,7
140 REM MODE 0 LINES
150 FOR X=7 TO 10:POKE DLIST+X,2:NEXT X
160 REM MODE 3 LINES
170 FOR X=11 TO 28:POKE DLIST+X,8:NEXT X
```

```
180 REM SET JUMP INSTRUCTION
190 POKE DLIST+29,65
200 REM BEGINNING OF DISPLAY LIST
210 POKE DLIST+30,PEEK(560)
220 POKE DLIST+31,PEEK(561)
```

Notice that a GRAPHICS 0 statement was used in line 15. This was needed to tell BASIC to open the screen and reserve some screen RAM. GRAPHICS mode 0 is used because it reserves the largest amount of RAM of any of the modes that will be used.

Enter the lines but don't run it yet. Let's wait until we have something to print on the screen.

Writing to a Mixed Mode Screen

Writing to a mixed mode screen can sometimes be difficult. Attempts to PRINT and PLOT to various portions of the screen are often interpreted incorrectly by BASIC. Since BASIC did not directly support the customized display list, it also does not directly support the normal methods for writing to the screen. It is, however, possible to change certain memory addresses to correct the problem. Two things must be done in order to consistently write to mixed mode displays.

1. First, you must instruct the computer as to which mode you are about to write. This can be done by POKEing address 87 with the BASIC GRAPHICS mode. For example, before printing to the BASIC mode 0 portion of the screen, we would use the following statement.

```
POKE 87,0
```

What statement would you use if you were writing to a BASIC mode 3 portion of the screen?

Answer #5 ______________

2. The next step is to tell the computer where the beginning of screen memory is for the mode window to which you are writing. First, we must find the screen memory address of the upper left hand corner of the screen. It can be calculated using the screen RAM pointers.

```
SCREENMEM = PEEK(88) + PEEK(89) * 256
```

Now calculate the amount of screen memory between the upper left hand corner of the screen and the upper left hand corner of the mode window to which you will be writing.

In this example, there are two lines of BASIC mode 2 above the BASIC mode 0 window. Refering to Table 7-1, note that each of these lines requires 20 bytes of RAM. So, the beginning of our BASIC mode 0 window will start at SCREENMEM + 40.

You must now POKE this new value back into addresses 88 and 89. But first you must break the value into high and low byte values. You could use these statements to calculate the new high and low bytes of the screen RAM address and put them back into the screen memory pointer addresses.

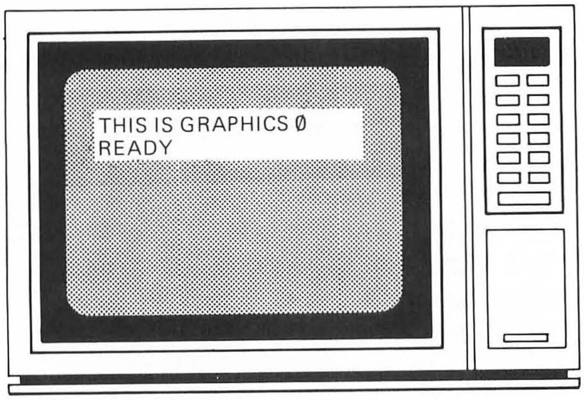
```
HI = INT(SCREENMEM + 40)/256)
LO = (SCREENMEM + 40) - (HI * 256)
POKE 89,HI
POKE 88,LO
```

Add these lines to your program:

#5. POKE 87,3

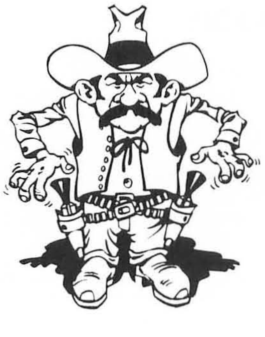
```
300 REM WRITE TO THE SCREEN
310 SCREENMEM=PEEK(88)+PEEK(89)*256
320 HI=INT((SCREENMEM+40)/256)
330 LO=(SCREENMEM+40)-(HI*256)
340 POKE 87,0
350 POKE 88,LO
360 POKE 89,HI
370 PRINT CHR$(125):REM CLEAR SCREEN
380 POSITION 0,0:PRINT "THIS IS GRAPHICS 0"
```

Press **SYSTEM RESET** to clear the screen. Run it. Your screen should look like this:



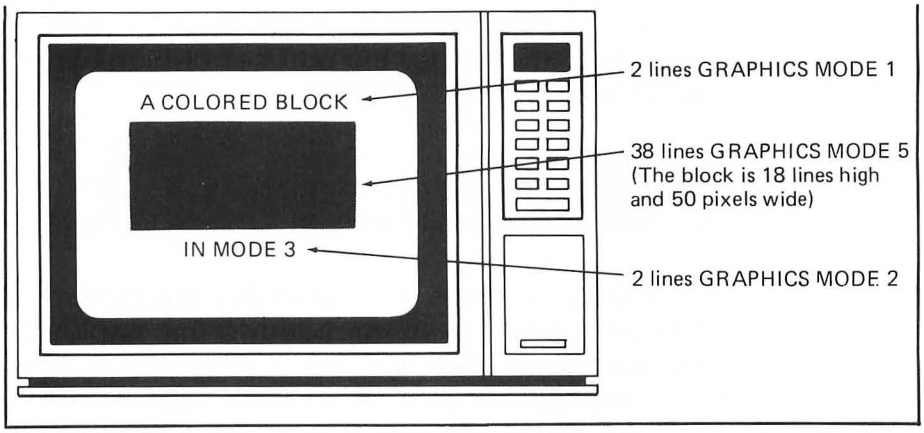
```
THIS IS GRAPHICS Ø
READY
```

Let's take time out for a challenge.



Challenge #12

Write a program to make the screen appear as shown below.


A COLORED BLOCK
IN MODE 3
2 lines GRAPHICS MODE 1
38 lines GRAPHICS MODE 5 (The block is 18 lines high and 50 pixels wide)
2 lines GRAPHICS MODE 2

Examples and Samples

When are mixed mode screens appropriate to use? An obvious answer is whenever you desire a screen that BASIC does not normally support. Here are some examples.

1. A page of text or a graphics diagram can be appropriately titled with a BASIC mode 1 or 2 mode line near the top of the screen.
2. BASIC GRAPHICS mode 0 windows can be placed appropriately on the screen.
3. A single graphics diagram can be composed of several GRAPHICS modes.
4. ANTIC modes 3, 4, 5, 12, and 14 are not normally supported by BASIC but can be used in custom display lists. Four color characters can be used with ANTIC modes 4 and 5 (more on this in Chapter 10). ANTIC mode 14 also uses four colors with the same vertical resolution as BASIC mode 8.
5. The screen size can be adjusted to include more than 192 scan lines.
6. Blank lines can be used to separate parts of the screen.
7. You can have multiple screens and multiple display lists.

Let's try samples of a couple of these examples. Here is a program that changes the number of blank lines at the top of the GRAPHICS mode 0 screen:

```
10 GRAPHICS 0
20 DLIST=PEEK(560)+PEEK(561)*256
30 POKE DLIST,0:POKE DLIST+1,0:POKE DLIST+2,0
```

Run it. Do you notice that the screen has moved up? The blank line codes have been changed in the display list. The blank line codes are: 0—1 line, 16—2 lines, 32—3 lines, 48—4 lines, 64—5 lines, 80—6 lines, 96—7 lines, 112—8 lines.

Let's try another example. Enter and run the following program:

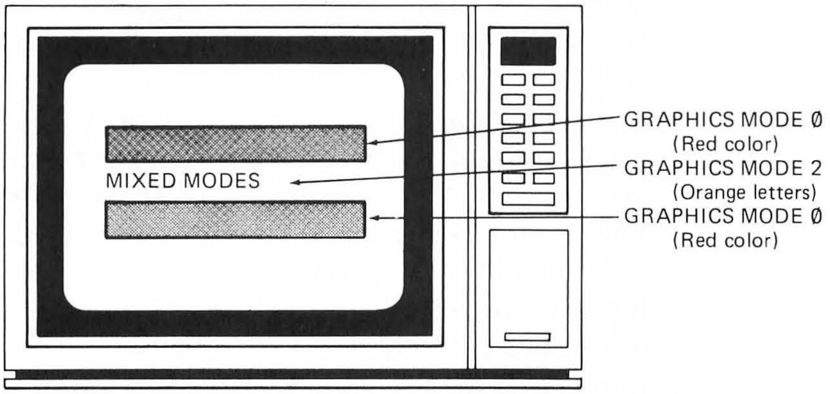
```
100 REM MULTIPLE SCREENS
200 REM SCREEN 1
210 MEMTOP=PEEK(106):REM TOP OF MEMORY
220 GRAPHICS 5
230 DLIST1=PEEK(560)+PEEK(561)*256:REM DISPLAY LIST 1
240 REM DRAW SOMETHING ON SCREEN 1
250 COLOR 1
260 FOR X=1 TO 79 STEP 2
270 PLOT X,0:DRAWTO X,39
280 NEXT X
290 PRINT "WHICH DISPLAY (1 OR 2)"
300 REM SCREEN 2
310 POKE 106,MEMTOP-8:REM MOVE TOP OF MEMORY DOWN
320 GRAPHICS 3
330 DLIST2=PEEK(560)+PEEK(561)*256:REM DISPLAY LIST 2
340 REM DRAW SOMETHING ON SCREEN 2
350 COLOR 2
360 FOR Y=1 TO 19 STEP 2
370 PLOT 0,Y:DRAWTO 39,Y
380 NEXT Y
400 REM FLIP BETWEEN SCREENS
410 PRINT "WHICH DISPLAY (1 OR 2)":INPUT D
420 IF D=1 THEN DLIST=DLIST1
430 IF D=2 THEN DLIST=DLIST2
500 REM CALCULATE HI AND LO BYTES OF DISPLAY LIST
510 HI=INT(DLIST/256):LO=DLIST-(HI*256)
520 POKE 561,HI:POKE 560,LO
530 GOTO 400
```

The program has set up two display lists and two screen RAM areas. Because we have sort of stranded the screen 1 RAM, we don't see our entry of a 1 or 2 appear in the text window when screen 1 is displayed. The program, however, will still respond to your input.

This technique provides us with an excellent method of changing between complex screen images without waiting to see them redrawn each time. The number of screens available is only limited by available RAM.

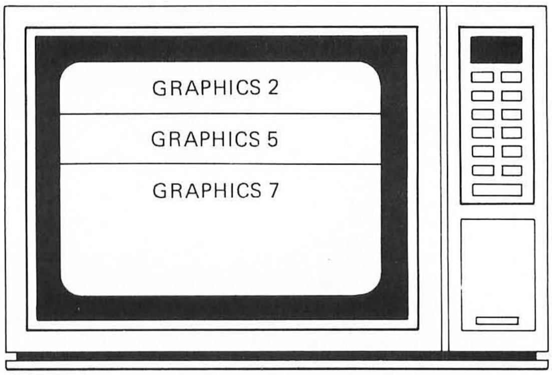
Challenge #13

Write a program to produce the following screen image.



Chapter 7 Review

Answer the following questions.
The following diagram is used for questions 1, 2, and 3.

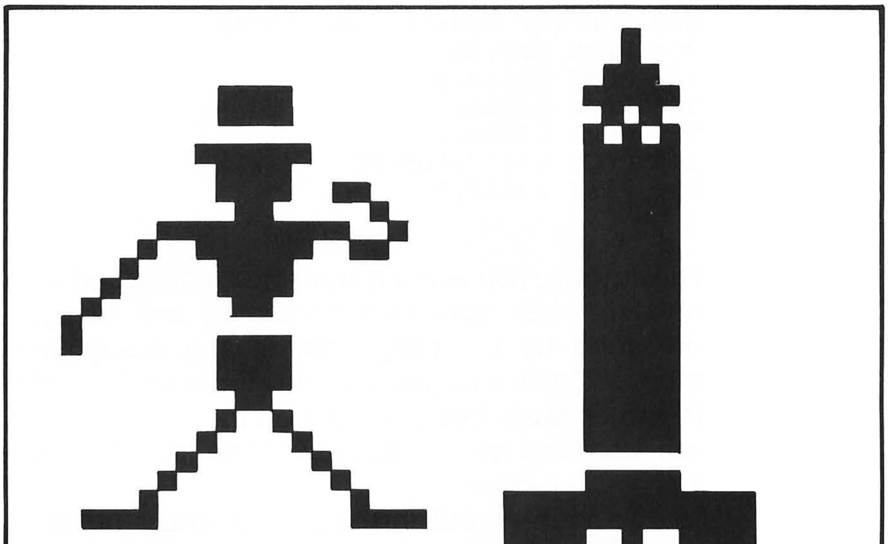


1. How many scan lines are required to produce four GRAPHICS 2 mode lines?
2. A screen of 192 scan lines is made up of BASIC GRAPHICS modes 2, 5, and 7. If you use 4 GRAPHICS 2 mode lines and 10 GRAPHICS 5 mode lines, how many mode lines remain for GRAPHICS 7?
3. If each of the three graphics mode windows occupy the same amount of screen space, how many mode lines would be required for each graphics mode window?
4. Which memory address is used to set the GRAPHICS mode before writing to the screen?
5. If the screen address has a decimal value of 14296, what are the values of the high and low bytes?
6. How many ANTIC mode bytes are required in the display list to display 80 scan lines of BASIC GRAPHICS mode 5?
7. Which addresses hold the value for the RAM location of the display list?
8. Write a display list of decimal values that would produce a screen of two mode lines of GRAPHICS 0 followed by the remainder of the screen in GRAPHICS 2.

CHAPTER 8

Playing with Players and Missiles

Players? Missiles? It sounds dangerous.



In this chapter you will learn to:

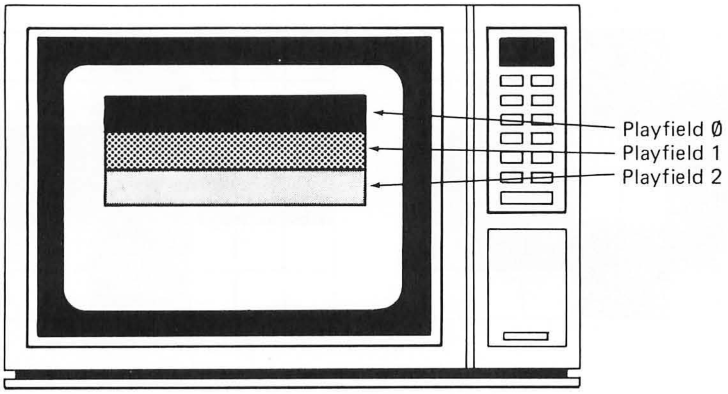
- Build players and missiles
- Move players and missiles
- Explain player/missile/playfield relationships

The ATARI computer has a special facility for using players and missiles. But what are players and missiles? The best answer to this question comes with a demonstration. Enter and run the following program:

```
100 REM PLAYER/MISSILE DEMONSTRATION
110 GRAPHICS 0
200 REM SET PLAYER BASE ADDRESS
210 PMBASE=PEEK(106)-8
220 REM CLEAR MEMORY AND PUT IMAGE DATA IN
230 FOR X=0 TO 255:POKE PMBASE*256+X,0:NEXT X
240 FOR X=40 TO 45:READ D:POKE PMBASE*256+512+X,D:NEXT X
250 DATA 24,24,60,60,126,126
300 REM SET PLAYER PARAMETERS
310 POKE 559,46
320 POKE 53248,100
330 POKE 53256,3
335 POKE 53256,3
340 POKE 54279,PMBASE
350 POKE 53277,3
```

The image you see on the screen is called a *player*. Players and missiles, generally speaking, are images that can be designed by the programmer and easily moved about the screen. These images are also independent of text and drawings that have been placed on the screen. List the program to see what we mean. The programs lines will be listed over the top of the player.

To understand this sort of graphics utility, we need to go back and review what we already know about ATARI graphics. In Part One of this book we learned to draw using the various color registers. Each time we used a particular register we were drawing what is called a *playfield*. For example, the color display program of Chapter 3 used three playfields because three color registers were used to draw the image on the screen.

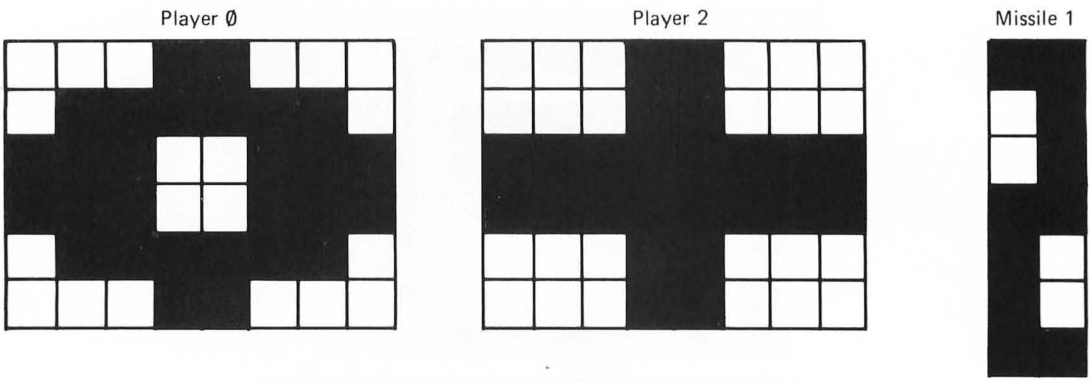


Each pixel in a playfield is represented by data which is stored in the screen memory portion of RAM. The upper left hand corner of the screen starts at a particular RAM location and the screen data is contained in subsequent addresses until the lower right hand corner of the screen is reached. Two distinct playfields cannot occupy the same portion of screen memory and, thus, playfields may not overlap. The different playfields are simply represented by different data values in screen memory depending on which color register is used to draw the playfield.

Players and missiles, on the other hand, occupy a separate section of memory and can be displayed on the screen independent of the playfield data.

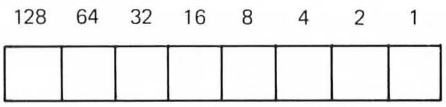
Building Players and Missiles

We can learn best by doing an example. A player is an image that is 8 bits(pixels) wide and may be up to 256 scan lines high. Four different players can be used at one time. A *missile* is an image that is 2 bits(pixels) wide and may be up to 256 scan lines high. Four different missiles can be used at one time. Let's design the shape of two players and a missile. We'll call them Player 0, Player 2, and Missile 1.

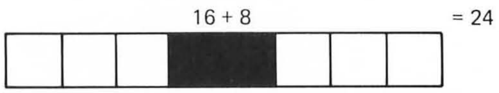


Each line of each player is eight bits wide and can be represented by a byte. Since we will eventually POKE this information into memory using BASIC, we will represent each of these lines(bytes) with a decimal value.

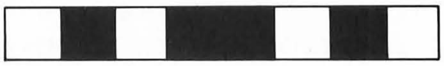
Let's look at Player 0 line by line. Each of the eight bits in a line can be represented by a decimal value as shown below.



If the two center bits are to be turned ON, as is the case with first line in Player 0, then the code representing this would be



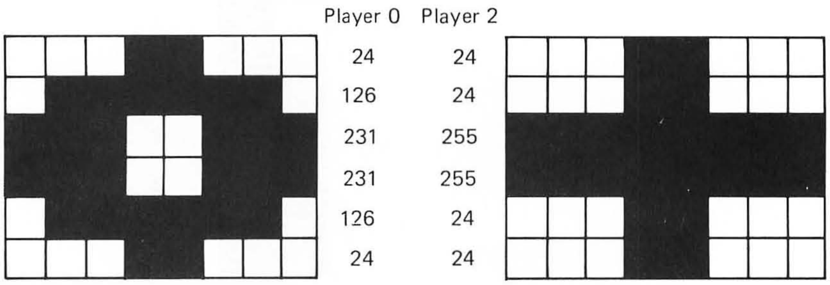
What would be the decimal equivalent for this line?



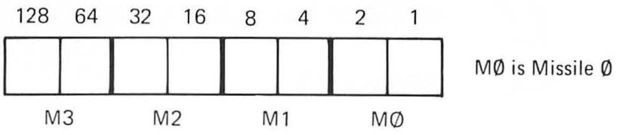
Answer #1 ____________

#1. 90

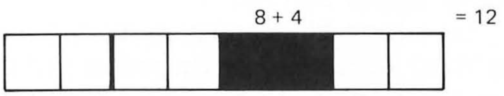
Each of the players can be represented with the following values:



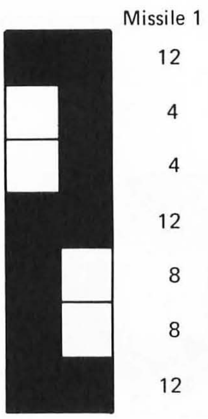
Let's find the values representing Missile 1. Each line of the missile is only two bits wide. However, data for all four missiles are encoded into the same byte, so the computation is somewhat different than it is for players.



The first line of Missile 1 would be



The entire missile is represented as follows.



Suppose we were using the same image for Missile 0 as we did for Missile 1 (Missile 1 not used). What would be the value of the byte representing the first line?

Answer #2 ______________

Suppose that we were using the same image as above for both Missile 1 and Missile 0. What would be the value of the byte representing the first line?

Answer #3 ______________

Now that we have the players and missile designed what will we do with them? Let's store them in memory. But they can't be stored just anywhere. Each player is mapped into either 128 or 256 bytes depending on whether single scan line or double scan line resolution is required. The four missiles also require a total of 128 or 256 bytes.

Let's look at the player/missile memory map in Figure 8-1. This map shows the arrangement of the players and missiles in memory relative to a player/missile base address (PMBASE). We will use this map to determine the storage locations for the players and missile we've designed.

#2. 3 #3. 15

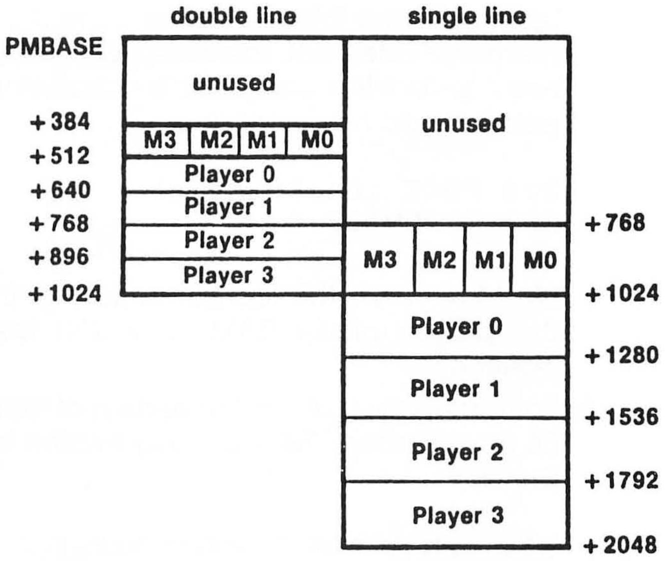


FIGURE 8-1. Player-Missile RAM Map. (Reprinted from the De Re Atari Guide by permission of Atari, Inc.)

Let's do the procedure for building players and missiles and write a program at the same time. We will build the players and missile using double line resolution. This means that each player or missile byte will map onto two scan lines on the screen.

1. Let's start with this line:

```
100 REM PLAYER 0,2 AND MISSILE 1
```

2. Next, we need to find an area of RAM that can be used without interference. Let's find the top of memory and back off four 256 byte pages or 1024 bytes. Eight pages are required for single line resolution. Enter the following statements:

```
200 REM SET PLAYER/MISSILE BASE ADDRESS
210 PMBASE=PEEK(106)-4
```

So that screen RAM, which is normally located at the top of memory, does not interfere with player/missile RAM, we need to fool the computer by changing the top of memory pointer. Add two more lines:

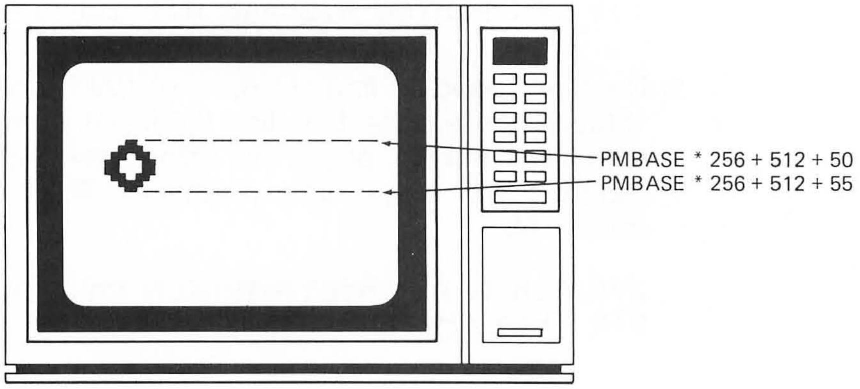
```
220 POKE 106,PMBASE-1
230 GRAPHICS 0
```

Line 220 resets the top of memory pointer one page below the player/missile RAM. Line 230 resets the screen RAM position.

3. To make sure that this top section of memory is clear, we can fill it with zeros. The following routine will do the trick. Enter the lines:

```
300 REM CLEAR PLAYER.MISSILE RAM
310 FOR X=384 TO 1024
320 POKE PMBASE*256+X,0
330 NEXT X
```

4. Now we can put our player and missile data into this area of RAM. The vertical position of a player depends on its position in memory. Each player may be placed anywhere within the 128 vertical positions on the screen by being mapped anywhere into the 128 bytes used for that particular player. For example, Player 0 can be mapped anywhere between PMBASE*256+512 and PMBASE*256+640. If we want Player 0 to appear beginning 50 double scan lines from the top of the TV screen, it will be mapped into memory beginning at location PMBASE*256+512+50.



The following routine could be used. Enter these lines:

```
400 REM READ PLAYER MISSILE DATA
410 FOR TIMES=1 TO 3
420 READ ADDR1,ADDR2
430 FOR X=ADDR1 TO ADDR2
440 READ N:POKE PMBASE*256+X,N
450 NEXT X
455 NEXT TIMES
460 REM PLAYER 0
465 DATA 562,567,24,126,231,231,126,24
470 REM PLAYER 2
475 DATA 818,823,24,24,255,255,24,24
480 REM MISSILE 1
485 DATA 434,440,12,4,4,12,8,8,12
```

Next, we need to provide ANTIC with such information as color, position of players, and so on. A complete listing of all player/missile memory addresses is given in Table 8-1 and will be helpful in determining specific memory addresses for each player.

5. To set the color of the players and missile, we need to POKE the color value into the player/missile color registers. Add the following lines.

```
500 REM SET COLORS
505 POKE 704,0
510 POKE 705,200
520 POKE 706,30
```

The values 704, 705, and 706 are the color register addresses of Player 0, Player 1, and Player 2, respectively. Missile 1 will be the same color as Player 1. The color values can be related to the colors of Table 1-1 with the following formula:

$$\text{Hue} * 16 + \text{Luminance} = \text{color value}$$

In this case the values for color calculate to

$0 * 16 + 0 = 0$	A hue of 0 is black
$12 * 16 + 8 = 200$	A hue of 12 is green
$1 * 16 + 14 = 30$	A hue of 1 is orange

What would be the color value of a pink (4) hue with a luminance of 10?

Answer #4 ________________

6. Each of the players' and missile's horizontal position is set with a horizontal position register. Enter these lines:

```
600 REM SET HORIZONTAL POSITION
610 POKE 53248,100
620 POKE 53250,160
630 POKE 53253,120
```

The values 53248, 53250, and 53253 are the horizontal position registers for Player 0, Player 2, and Missile 1, respectively. Each register can hold a value from 0 to 228. Some of these position values, however, would be off of the edge of the screen.

7. The vertical resolution is set for double scan lines. Type these lines.

```
700 REM SET VERTICAL RESOLUTION
710 POKE 559,46
```

The value 559 is a shadow address for the direct memory access control. Bit 4 controls the vertical resolution. The value 46 is POKED into this address for double line resolution and 64 is used for single line resolution.

8. Finally, enable the players. Type these lines:

```
800 REM ENABLE THE PLAYERS
810 POKE 54279,PMBASE
820 POKE 53277,3
```

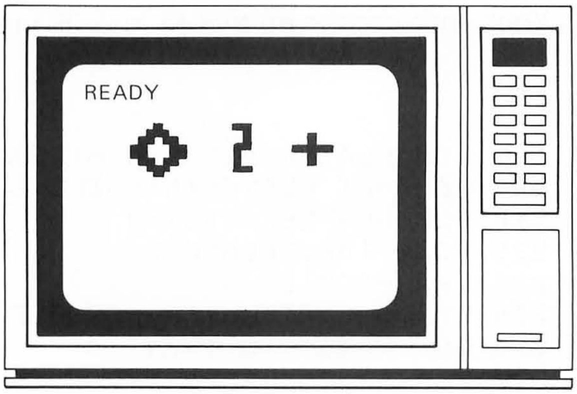
The value 54279 is the register that stores the value of the base address for the players and missiles. Address 53277 is the player missile enable register.

#4. 74

Your entire program should look like this:

```
100 REM PLAYER 0,2 AND MISSILE 1
200 REM SET PLAYER/MISSILE BASE ADDRESS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 0
300 REM CLEAR PLAYER.MISSILE RAM
310 FOR X=384 TO 1024
320 POKE PMBASE*256+X,0
330 NEXT X
400 REM READ PLAYER MISSILE DATA
410 FOR TIMES=1 TO 3
420 READ ADDR1,ADDR2
430 FOR X=ADDR1 TO ADDR2
440 READ N:POKE PMBASE*256+X,N
450 NEXT X
455 NEXT TIMES
460 REM PLAYER 0
465 DATA 562,567,24,126,231,231,126,24
470 REM PLAYER 2
475 DATA 818,823,24,24,255,255,24,24
480 REM MISSILE 1
485 DATA 434,440,12,4,4,12,8,8,12
500 REM SET COLORS
505 POKE 704,0
510 POKE 705,200
520 POKE 706,30
600 REM SET HORIZONTAL POSITION
610 POKE 53248,100
620 POKE 53250,160
630 POKE 53253,120
700 REM SET VERTICAL RESOLUTION
710 POKE 559,46
800 REM ENABLE THE PLAYERS
810 POKE 54279,PMBASE
820 POKE 53277,3
```

Run it. It takes a few seconds to run so be patient. Does your screen look something like the following?



About Players and Missiles

Let's discover more about players and missiles by making some changes to our program. Refer to Table 8-1 as you try each of the following changes.

Vertical Position

Move Missile 1 down the screen by changing its position in memory. You can shift the position by changing the value in the DATA statement on line 485. Change line 485 to

```
485 DATA 464,470,12,4,4,12,8,8,12
```

Press **SYSTEM RESET** to set the top of memory pointer back to normal. Now run it. Did Missile 1 move down on the screen?

In the next chapter we will animate a player by moving data through memory.

TABLE 8-1. Player/Missile Memory Addresses*

Description	Address
Direct Memory Access (DMA) Control	559
Graphic Control	53277
Graphics for all Missiles	53265
Graphics for Player 0	53261
Graphics for Player 1	53262
Graphics for Player 2	53263
Graphics for Player 3	53264
Horizontal Position of Missile 0	53252
Horizontal Position of Missile 1	53253
Horizontal Position of Missile 2	53254
Horizontal Position of Missile 3	53255
Horizontal Position of Player 0	53248
Horizontal Position of Player 1	53249
Horizontal Position of Player 2	53250
Horizontal Position of Player 3	53251
Color-Luminance of Background	712
Color Luminance of Playfield 0	708
Color Luminance of Playfield 1	709
Color Luminance of Playfield 2	710
Color Luminance of Playfield 3	711
Color Luminance of Player—Missile 0	704
Color Luminance of Player—Missile 1	705
Color Luminance of Player—Missile 2	706
Color Luminance of Player—Missile 3	707
Sizes for all missiles	53260
Size of Player 0	53256
Size of Player 1	53257
Size of Player 2	53258
Size of Player 3	53259
Priority Select	623
Player Missile Base Address	54279

*Adapted from the De Re Atari Guide by permission of Atari, Inc.

Horizontal Position

Change the horizontal position of Missile 1. This is easily done by changing the value in the horizontal position register. You can do it in immediate mode. Type

```
POKE 53253,160 RETURN
```

Did Missile 1 move under Player 2? You can of course set these positions in program mode as well. The value of this register can range from 0 to 228. However, some of these position values are off the edge of the screen. If you want to make a player or missile disappear you can set its horizontal position registers to 0.

?

If the screen is 228 units wide, what value would you use to set a player to the horizontal center of the screen?

Answer #5 ________________

Width

The width of players or missiles can be easily changed. Again, let's try it in immediate mode. Type

```
POKE 53260,1 RETURN
```

Missile 1 should now have double width. Let's quadruple the size. Type

```
POKE 53260,3 RETURN
```

Refer to the player/missile memory addresses in Table 8-1 for the address of the other players. The width can be set back to normal by POKEing a 0 into address 53260.

#5. 114

What statement would you use to change the width of Player 0 to quadruple size?

Answer #6 ________________

Priorities

Suppose the horizontal positions of Player 0 and Player 2 were both set to the same value. Which of the two players would appear to be on top? Let's try it. Set the horizontal position of both players to 160. Type

```
POKE 53248,160 RETURN
```

Can you see Player 2 showing through the center of Player 0? Change the priority by typing this statement:

```
POKE 623,2 RETURN
```

Notice that Player 2 has disappeared.

Priorities can be established between players or between players and playfields. Figure 8-2 shows priorities and the corresponding bits of the priority register (PRIOR) which must be set.

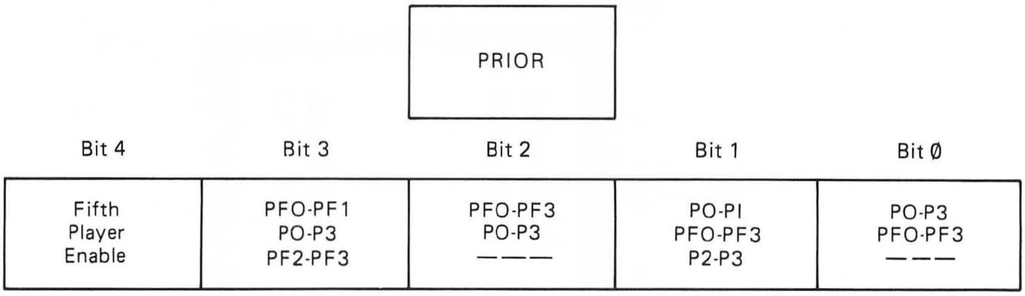


FIGURE 8-2. Player/Playfield Priorities

#6. POKE 53256,3

In our example, the statement POKE 623,2 (Bit 1 set) gave Player 0 and Player 1 (P0-P1) priority over all playfields (PF0-PF3). These in turn have priority over Player 2 and Player 3 (P2-P3). Since the background had a higher priority than Player 2, Player 2 disappeared behind the background.

What statement would you use to give Player 0 (P0) priority over Playfield 3 (PF3)?

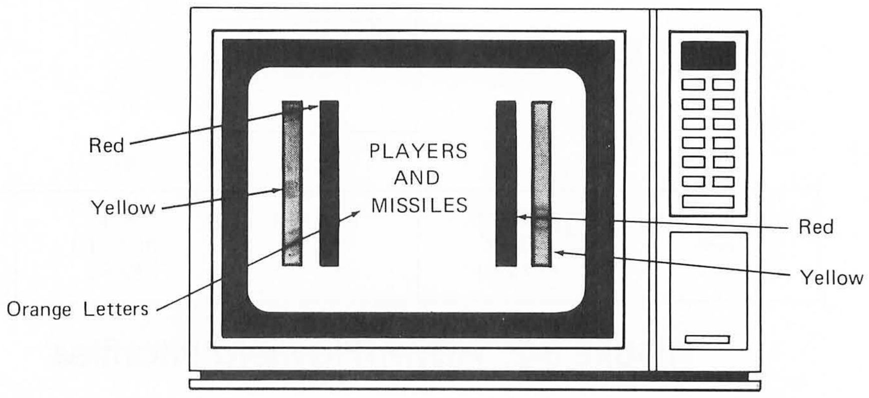
Answer #7 ________________

The four missiles can also be combined into a fifth player. If bit 4 is set in PRIOR, the fifth player takes the color value of playfield color register 3. The positions are still set independently as missiles.

Challenge #14

Write a program that will make the screen look like this. Use GRAPHICS mode 1 for the text.

Hint: The red and yellow bars are players.



#7. POKE 623,1

Collisions

Hardware registers also exist to detect overlapping or collision of players with players or missiles and players with playfields. Table 8-2 shows hardware collision detection registers.

Let's try an example. Press **SYSTEM RESET** and run the program again. To see if Player 0 has collided with another player, check the value of its collision register. Type

```
PRINT PEEK(53260) RETURN
```

It should print the value 0 indicating that no collision has taken place. Now make them collide. Type

```
POKE 53248,160 RETURN
```

TABLE 8-2. Collision Detection Addresses

Description	Address
Missile 0 to Playfield Collisions	53248
Missile 0 to Player Collisions	53256
Missile 1 to Playfield Collisions	53249
Missile 1 to Player Collisions	53257
Missile 2 to Playfield Collisions	53250
Missile 2 to Player Collisions	53258
Missile 3 to Playfield Collisions	53251
Missile 3 to Player Collisions	53259
Player 0 to Playfield Collisions	53252
Player 0 to Player Collisions	53260
Player 1 to Playfield Collisions	53253
Player 1 to Player Collisions	53261
Player 2 to Playfield Collisions	53254
Player 2 to Player Collisions	53262
Player 3 to Playfield Collisions	53255
Player 3 to Player Collisions	53263
Collision Clear	53278

Check the collision register again. Type

```
PRINT PEEK(53260) RETURN
```

The nonzero value that is printed indicates that a collision has occurred. To clear the collision register of any collision, POKE any value into address 53278 (collision clear register).

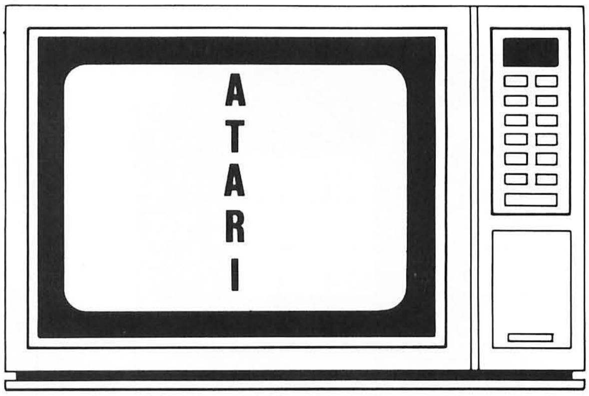
Collision detection is primarily useful for animated screens. We'll study animation in Chapter 9.

Applications

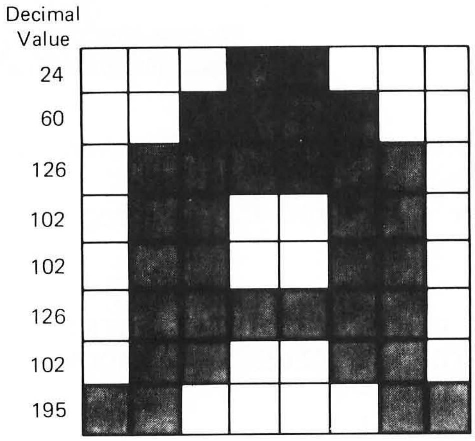
The number of ways to creatively use players and missiles is only restricted by your imagination. Here are a few ideas:

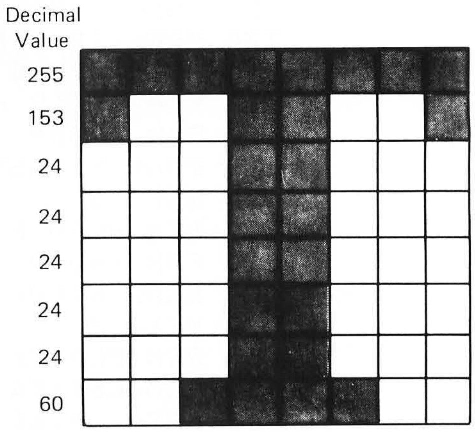
1. Players can be used to add four additional colors to your display and they don't depend on the GRAPHICS mode you are using.
2. You can use players to print characters vertically in any GRAPHICS mode.
3. Players can be easily animated (see Chapter 9).
4. Players can be made to be transparent and can, therefore, be used to highlight words or parts of diagrams.
5. Players can be used with menus to show positions for making a selection.
6. Three dimensional motion can be achieved by setting appropriate priorities and adjusting player width.

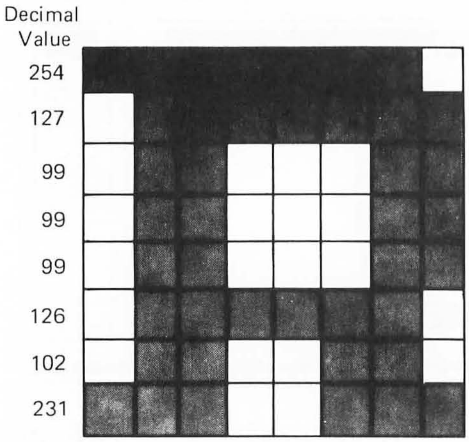
Let's try an example. We will use a player to produce some characters and display them vertically on the screen as shown below.

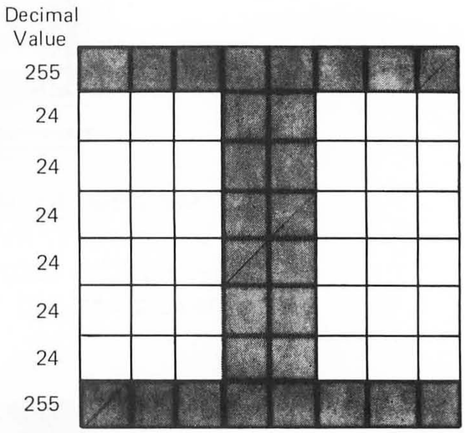


First, we need to design the characters that we'll use. They are the letters A, T, R, and I. Make your own design or use the following. The character design worksheet in Appendix A can be used as an aid.









These values can be read from DATA statements and POKED into the player/missile memory.

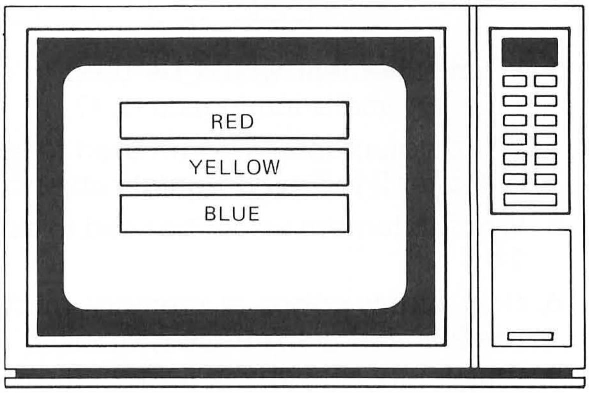
Only one player is required for this program. Enter the program and try it.

```
100 REM USING A PLAYER FOR CHARACTERS
200 REM SET PLAYER/MISSILE BASE ADDRESS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 0
300 REM CLEAR PLAYER/MISSILE RAM
310 FOR X=384 TO 1024
320 POKE PMBASE*256+X,0
330 NEXT X
400 REM READ PLAYER MISSILE DATA
410 FOR X=540 TO 588
420 READ N:POKE PMBASE*256+X,N
430 NEXT X
450 REM PLAYER 0 DATA
455 REM LETTER A
460 DATA 24,60,126,102,102,126,102,195,0,0
```

```
465 REM LETTER T
470 DATA 255,153,24,24,24,24,24,60,0,0
475 REM LETTER A
480 DATA 24,60,126,102,102,126,102,195,0,0
485 REM LETTER R
490 DATA 254,127,99,99,99,126,102,231,0,0
495 REM LETTER I
497 DATA 255,24,24,24,24,24,24,255,0,0
500 REM SET COLORS
510 POKE 704,31
600 REM SET HORIZONTAL POSITION
610 POKE 53248,114
700 REM SET VERTICAL RESOLUTION
710 POKE 559,46
800 REM ENABLE PLAYER 0
810 POKE 54279,PMBASE
820 POKE 53277,3
```

Challenge #15

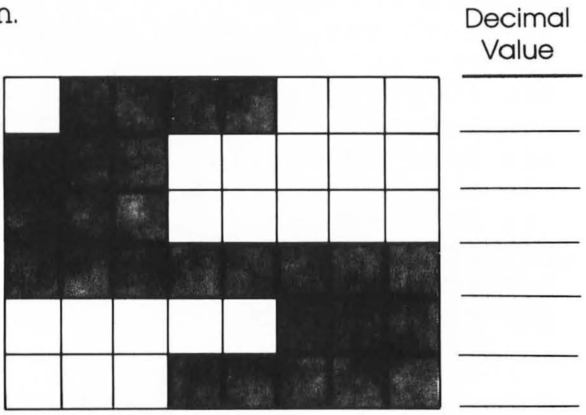
Write a program to produce the following display in GRAPHICS mode 0. Three players (red, yellow, and blue) are quadruple width and are placed over the graphics mode 0 text.



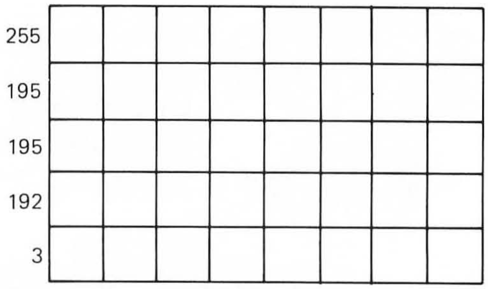
Chapter 8 Review

Answer the following questions.

1. What are the decimal values for each line of the following diagram.



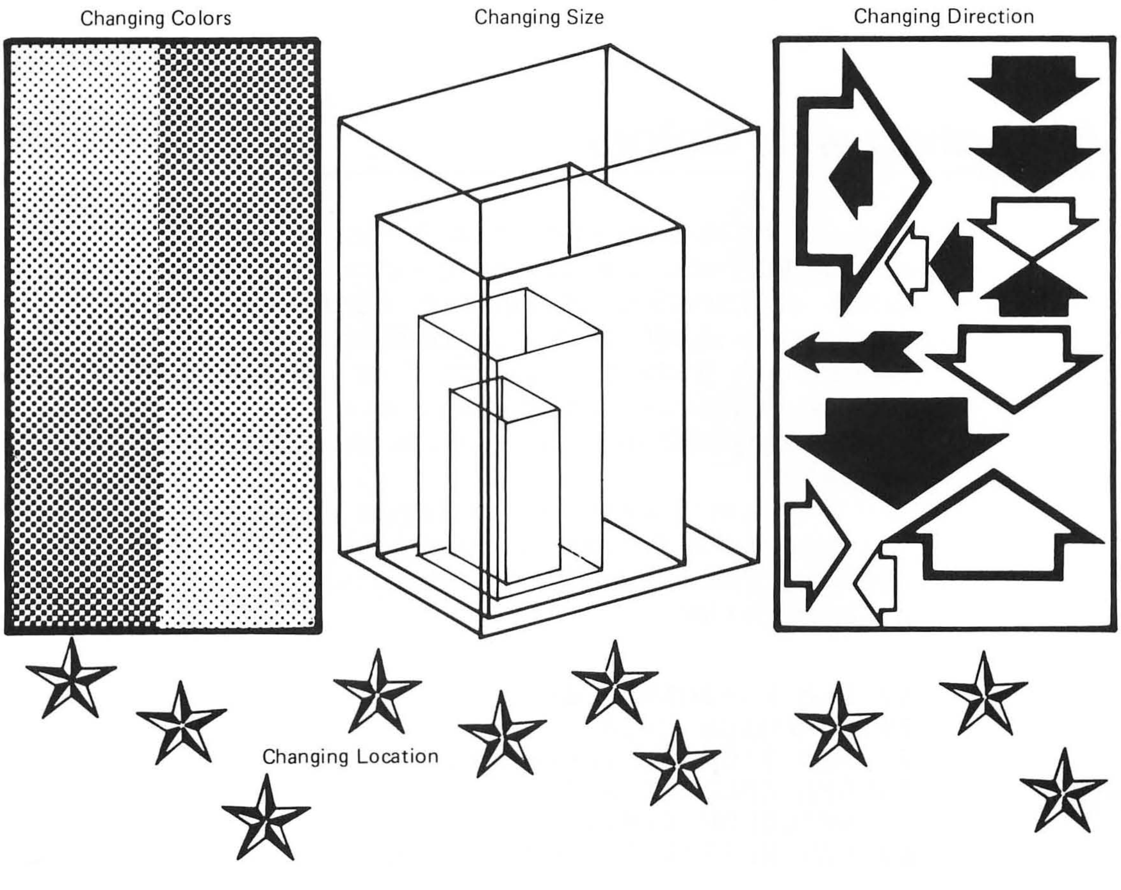
2. Draw the shape of the player defined by the five bytes listed.



3. What statement would be used to set Player 2 to a hue of blue (8) and a luminance of 4?
4. What statement would be used to set the horizontal position of Player 2 to the horizontal center of the screen?
5. What statement would be used to double the width of Player 2?
6. How many pages of memory must be reserved for single resolution player/missile graphics?
7. Which bit of the PRIOR register must be set to give Playfield 0 priority over Player 3?

CHAPTER 9
Animation

Animation is change.



In this chapter you will learn to:

- Animate characters
- Animate playfields
- Animate players and missiles
- Create animated screens

If you have been to Disneyland or Disneyworld you have probably witnessed animation first hand. Or, perhaps you've seen animated cartoon characters on Saturday morning television. We will use the word *animation* to mean more than that. We will consider a screen image that undergoes any change at all to be animated. It could be flashing characters or, perhaps, changing border colors or the movement of images or characters.

Animating with Colors

Using colors to draw attention to the screen or screen images can be very effective. Changing colors can alert the user to errors or important information. Border colors might be changed to signify a change of modes or operations in a program. A flashing background or character color may provide a warning. Colors might even be rotated through various playfields to simulate movement. Let's try a couple of examples.

In Chapter 5 we learned to change the colors of characters. We can make characters appear to flash on and off by changing their colors. Here is a demonstration program. Enter the lines and try it:

```
10 REM FLASHING TEXT
20 GRAPHICS 2+16
30 POSITION 8,5:PRINT #6;"FLASH"
40 REM COLOR CHANGE
50 SETCOLOR 0,0,14
60 FOR WAIT=1 TO 100:NEXT WAIT
```

```
70 SETCOLOR 0,0,0
80 FOR WAIT=1 TO 100:NEXT WAIT
90 GOTO 50
```

The colors of the characters are alternated between white and black (background color).

Could you make the characters flash between pink and blue? Which two statements would you change?

Answer #1 ________________
Answer #2 ________________

Changing colors can also be used to simulate motion. Try this example:

```
10 REM MOVING COLORS
20 GRAPHICS 3+16
30 REM DRAW 3 PLAYFIELDS
40 FOR X=1 TO 3
50 COLOR X
60 FOR J=1 TO 5
70 PLOT 5*X+5+J,5:DRAWTO 5*X+5+J,20
80 NEXT J
90 NEXT X
100 REM CHANGE COLORS IN SEQUENCE
110 FOR X=708 TO 710
120 POKE X,14
130 FOR WAIT=1 TO 100:NEXT WAIT
140 POKE X,0
150 NEXT X
160 GOTO 100
```

In this example, we have drawn three playfields on the screen. Two of the playfields have been colored black (background color) and the remaining playfield has been colored white. The playfield colors are changed between black and white to

#1. 50 SETCOLOR 0, 8, 6 #2. 30 SETCOLOR 0, 4, 6 make it appear as though a white rectangle moves about the screen. The three playfields have not moved; only the colors have changed.

If you have a GTIA chip, the results are more dramatic. Change these lines to see the effect:

```
20 GRAPHICS 10
40 FOR X=0 TO 8
60 FOR J=1 TO 8
70 PLOT 8*X+J,5:DRAWTO 8*X+J,170
110 FOR X=705 TO 713
```

Character Animation

So far we have only animated the screen by changing colors. Animation can also be achieved by the movement of characters. Let's begin by choosing a character and moving it across the screen. Enter and run the following program:

```
10 REM MOVE A CHARACTER
20 GRAPHICS 0
30 FOR P=0 TO 37
40 PRINT CHR$(0);
50 FOR WAIT=1 TO 100:NEXT WAIT
60 NEXT P
```

You should see the graphics character for a heart being slowly printed across the screen. But, suppose that we want to move a single character across the screen. We would need to erase the previous character each time a new character was printed. Try the following example:

```
10 REM MOVE A CHARACTER
20 GRAPHICS 0
30 FOR P=0 TO 37
40 POSITION P+1,0:PRINT CHR$(0)
50 POSITION P,0:PRINT CHR$(32)
60 FOR WAIT=1 TO 100:NEXT WAIT
70 NEXT P
```

In this routine, each time the printing position is moved one character to the right, a blank character CHR$(32) is printed into the previous position. The result is the apparent movement of the character across the screen.

Which two lines would you change to move the character vertically?

Answer #3 ________________
Answer #4 ________________

Character animation can also be accomplished by moving a character code through the screen RAM. Try this example:

```
10 REM MOVE CHARACTER IN SCREEN RAM
20 GRAPHICS 0
30 SCREENMEM=PEEK(88)+PEEK(89)*256
40 FOR P=0 TO 37
50 POKE SCREENMEM+P+1,64
60 POKE SCREENMEM+P,0
70 FOR WAIT=1 TO 100:NEXT WAIT
80 NEXT P
```

The beginning of the screen RAM is calculated and a character value is poked into memory. Each time the character is written one byte forward, it is erased from the previous memory location. Notice that the ASCII codes are not used to represent the characters that are placed on the screen. An internal code of 64 (heart character) and 0 (space) are used in lines 50 and 60 of our program. These internal character codes will be discussed in Chapter 10.

A character can be moved vertically on the screen as well. The screen memory begins at the upper left hand corner of the screen and is arranged so that the address increases as you move across the screen. After 40 screen characters (40 bytes of screen RAM), the characters move to the next line. To move a

#3. 40 POSITION 0, P + 1: PRINT CHR$(0) #4. 50 POSITION 0, P:PRINT CHR$(32)

character vertically on the screen, we need to move the character in increments of 40 bytes through memory. Let's try it. Change the following lines and run the program again.

```
40 FOR P=0 TO 960 STEP 40
50 POKE SCREENMEM+P,128
60 POKE SCREENMEM+P-40,0
```

Challenge #16

Write a program, using the POSITION statement, to move the number 0 across the screen in GRAPHICS mode 2.

Playfield Animation

Animating a playfield is similar to character animation. Let's try a demonstration in GRAPHICS mode 3 to animate a single pixel just as we animated a single character in GRAPHICS mode 0. Enter the program and run it.

```
10 REM MOVE A MODE 3 PIXEL
20 GRAPHICS 3
30 FOR P=0 TO 37
40 COLOR 1:PLOT P+1,0
50 COLOR 0:PLOT P,0
60 FOR WAIT=1 TO 100:NEXT WAIT
70 NEXT P
```

For each COLOR 1 pixel that is plotted, a COLOR 0 (background color) pixel is plotted into the previous position.

Which statement in the above program would you change to shorten the horizontal distance that the pixel appears to move?

Answer #5 ________________

Let's animate a larger playfield area. First we will draw a colored square in GRAPHICS mode 7. Then we will move it from the left side of the screen to the right side. Enter this program and run it.

```
10 REM PLAYFIELD ANIMATION IN MODE 7
20 GRAPHICS 7
30 REM DRAW AN IMAGE
40 FOR X=5 TO 10
50 PLOT X,45:DRAWTO X,50
60 NEXT X
100 REM MOVE IMAGE
110 FOR X=5 TO 154
120 COLOR 0:PLOT X,45:DRAWTO X,50
130 COLOR 1:PLOT X+5,45:DRAWTO X+5,50
140 NEXT X
```

To move the image we use the same technique we did with the single pixel. For each new vertical line we draw to the right of the image, we erase the line on the left side of the image. Again, this is accomplished by drawing COLOR 1 lines on the right side and COLOR 0 (background color) lines on the left side.

Notice that we have not included a wait loop in the program to slow it down. As the number of pixels to be moved increases, the slower the speed of animation becomes. Let's

#5. 30 for P = 0 to 20 (answers may vary) make the image larger and observe the change of speed as it moves from left to right. Change these lines and run it again.

```
 50 PLOT X,5:DRAWTO X,75
120 COLOR 0:PLOT X,5:DRAWTO X,75
130 COLOR 1:PLOT X+5,5:DRAWTO X+5,75
```

The animation we have done so far has been very simplified. The movement of the pixels of an irregular shape becomes more complicated and requires more computer time. Two dimensional animation also adds to the complexity of the program routine. Because of the speed limitations of BASIC, good playfield animation routines often require machine language programming.

Player Animation

The animation of some screen images can be very much simplified through the use of player/missile graphics. For example, horizontal movement is accomplished by simply changing the value in a horizontal position register. Let's build a player and demonstrate. Enter the following program:

```
100 REM PLAYER/MISSILE ANIMATION
200 REM BUILD PLAYER AND SET REGISTERS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 3+16
240 FOR X=512 TO 640
250 POKE PMBASE*256+X,0
260 NEXT X
270 FOR X=562 TO 570
280 POKE PMBASE*256+X,255
290 NEXT X
300 POKE 704,30
310 POKE 559,46
320 POKE 54279,PMBASE
330 POKE 53277,3
```

Run the program. A small yellow square should appear in the center of the screen.

The player can be animated by changing the values in the horizontal position register. Add the following routine to your program to make the square bounce back and forth from side to side:

```
400 REM ANIMATE PLAYER
405 REM MOVE TO RIGHT
410 FOR X=40 TO 208
420 POKE 53248,X
430 NEXT X
435 REM MOVE TO LEFT
440 FOR X=208 TO 40 STEP -1
450 POKE 53248,X
460 NEXT X
470 GOTO 400
```

Run it. It's not only smooth but it is fast as well. We can make it move even faster by changing the horizontal position register in increments of three. Try it. Change these lines:

```
410 FOR X=40 TO 208 STEP 3
440 FOR X=208 TO 40 STEP -3
```

Run the program again. The square should move faster but perhaps not as smoothly.

Now, let's animate it vertically. To do this we must use the same technique we used in playfield animation. We will move the player data either up or down in memory. If we move each of the player image lines up one byte in memory then we will need to erase the bottom player image line by placing a zero into that memory location. The same is true when the player data is moved down. Let's try an example. Add these lines to your program. Note that some of these additions are intended to change the lines.

```
400 REM ANIMATE PLAYER
405 REM MOVE DOWN
410 POKE 53248,114
420 FOR Y=503 TO 631
```

```
430 POKE PMBASE*256+Y,0
435 POKE PMBASE*256+Y+9,255
440 NEXT Y
450 REM MOVE UP
460 FOR Y=631 TO 503 STEP -1
470 POKE PMBASE*256+Y+9,0
480 POKE PMBASE*256+Y,255
490 NEXT Y
500 GOTO 400
```

Run the program. Does the player bounce between the top and bottom of the screen?

Challenge #17

Write a program to bounce two players, shaped like squares, at different speeds horizontally between the two sides of the screen.

Applications

Screen animation adds an entirely new dimension to your video compositions. Dynamic displays hold the viewer's attention. There are many ways that animation can be used. Here are a few:

1. Flashing colors attract the user's attention toward or away from points of interest on the screen.
2. Moving images have special application in games. The movement of these images might be controlled by joysticks or paddles.

3. Animation can be used to add additional information to a screen display. For example, a ball shown on the screen could be bouncing, rolling, or perhaps not be moving at all.
4. Animation might be used to show a process such as a container being filled with a liquid.

The uses are numerous. Each time you create a screen you should ask yourself if animation will appropriately add important information to the screen.

Let's try an example. We'll call it "The Closing Curtains." Enter and run the following program:

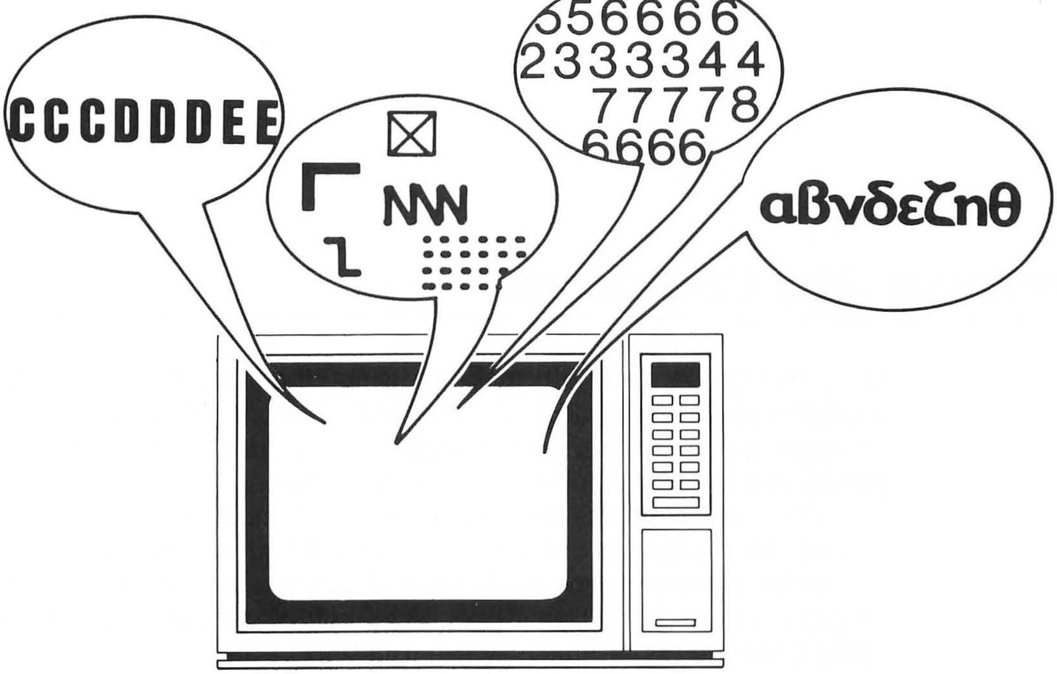
```
100 REM THE CLOSING CURTAINS
200 REM SET UP GRAPHCIS SCREEN
210 GRAPHICS 7+16
220 SETCOLOR 0,0,14
230 COLOR 1
300 REM DRAW VERTICAL LINES TOWARD CENTER
310 FOR X=0 TO 79
320 PLOT X,0:DRAWTO X,91:PLOT 159-X,0:DRAWTO 159-X,91
330 SETCOLOR 0,0,(90-X)/6
350 NEXT X
400 GOTO 400
```

Perhaps with a little more work the curtains could be made to look a little more realistic.

CHAPTER 10

New Character Sets

Would you like to create your own characters and display them on the TV screen?



In this chapter you will learn to:

- Create new character sets
- Create multicolor characters
- Print characters in any GRAPHICS mode
- Use multiple character sets

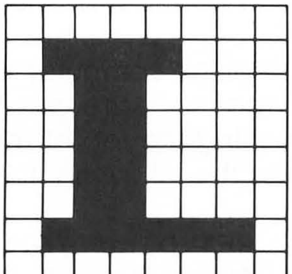
The graphics capability of any computer is enhanced by its ability to display a variety of different character types. These may be special graphics characters, special foreign language characters, or even an entire new font or type style for all of the characters in the set.

The ATARI computer contains a predetermined set of characters in a ROM known as the *character generator.* These characters are a part of the computer hardware and cannot be changed. However, it is not required that these predetermined characters be used. Instead, new characters can be generated, placed in RAM, and the computer instructed to use the new character set.

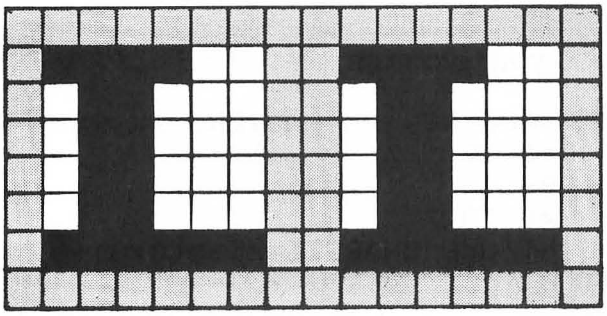
Defining New Characters

Let's design a new character using the character design worksheets in Appendix A. We will design the letter L but change the style somewhat from the normal upper case L produced by the ATARI character generator.

Each character is formed on an 8 × 8 matrix or grid. Each line of the character can be represented with a decimal value just as the players were in Chapter 8. Since a character consists of eight lines, eight bytes are required to define it. Our new letter L would be defined by the following values.

Character Image	Binary Code		Decimal Value
	0 0 0 0 0 0 0 0	=	0
	0 1 1 1 1 0 0 0	=	120
	0 0 1 1 0 0 0 0	=	48
	0 0 1 1 0 0 0 0	=	48
	0 0 1 1 0 0 0 0	=	48
	0 0 1 1 0 0 0 0	=	48
	0 1 1 1 1 1 1 0	=	126
	0 0 0 0 0 0 0 0	=	0

Notice that in the above example, the parts of the grid around the borders were not used. If they were, two characters printed next to each other would touch. This may or may not be desirable depending on the intended use of the character set.



The shaded part is not used unless you want characters to touch each other.

Character Sets

A full character set can have 128 characters. Each of these can be displayed in both normal and inverse video. Obviously, it is quite a lot of work to define an entire new character set.

Let's do a simple example. We'll build a set of 128 characters but make all of the characters in the set the same. We can use the character L that we've just defined.

1. The first step is to find a place in RAM to put the new set of characters. We can put it near the top of memory just as we

did with the players and missiles of Chapter 8. Since a character definition requires eight bytes, 128 characters would require 1024 bytes of RAM. This is equivalent to four 256 byte pages.

Again we need to fool the computer so that it places the screen RAM below the space we are using for our character set. This is done by resetting the top of memory pointer. To start your program, enter the following lines:

```
100 REM CHARACTER SET DEMONSTRATION
200 REM RESERVE CHARACTER SET RAM
210 CHARSET=PEEK(106)-4
220 POKE 106,CHARSET-1
230 GRAPHICS 0
```

CHARSET is the character base address expressed in 256 byte pages.

2. The next step is to write our new character into RAM. We can put the character definition bytes into DATA statements. Add these lines to your program:

```
300 REM PLACE NEW CHARACTERS IN RAM
310 FOR C=0 TO 127
320 RESTORE
330 FOR B=0 TO 7
340 READ N:POKE CHARSET*256+(C*8)+B,N
350 NEXT B
360 NEXT C
380 DATA 0,120,48,48,48,48,126,0
```

Notice that CHARSET is multiplied by 256 bytes/page to calculate the memory address of the new character set.

3. Finally, all we need to do is tell ANTIC where the character set base address is located. Add these lines:

```
400 REM SET CHARACTER SET POINTER
410 POKE 756,CHARSET
```

Run the program. (It takes about 30 seconds.) You should see an entire screen of L's. We have even turned the space character into an L. Let's change the space character back to normal. Press **SYSTEM RESET** and then add and change these lines:

```
301 FOR X=0 TO 7
302 POKE CHARSET*256+X,0
303 NEXT X


310 FOR C=1 TO 127
```

Now run it again. Press any key on the keyboard that you like. We have made a character set of L's. The characters printed on the keys no longer have the same meaning.

Changing Character Sets

In our last example, we wrote the character definition bytes for the space character into our character set RAM. But how did we know where the space character was located in the character set?

Each of the keys on the keyboard has a keyboard code (internal code) associated with it. When you press a key, the computer locates the character in the character set that corresponds to that key and prints the character on the screen. If you change the character image for that keyboard code, a different than normal character image is printed.

The keyboard code value is equivalent to the position of the keyboard character in the character set. A keyboard code of 0 would be the first character in the set and a keyboard code of 127 would be the last character in the set.

This keyboard code value can be calculated from the ASCII value (Table 5-1). The following table shows the computation that must be made to convert from ASCII to keyboard code.

ATARI ASCII value	Operation
0 to 31	Value + 64
32 to 95	Value − 32
96 to 127	None

Since the space character has an ASCII value of 32, the keyboard code is 32−32=0.

It's time for an example. Let's copy the ATARI character ROM into RAM so that we can modify one of the characters. Erase any old programs and enter these lines:

```
100 REM MODIFY ATARI CHARACTER SET
200 REM RESERVE CHARACTER SET RAM
210 CHARSET=PEEK(106)-4
220 POKE 106,CHARSET-1
230 GRAPHICS 0
300 REM COPY ATARI CHARACTER SET
310 FOR X=0 TO 1023
320 POKE CHARSET*256+X,PEEK(224*256+X)
330 NEXT X
```

An area of RAM is reserved just as before. The ATARI character ROM is located beginning at page 224. All 1024 character bytes are read from ROM and POKED into RAM.

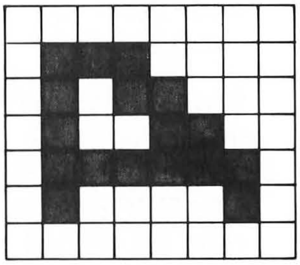
Now let's change the upper case T to a space. The eight character definition bytes for a space would all be zeros. To find the keyboard code for T, first you need to find the ASCII value (refer to Table 5-1). The ASCII value for T is 84 so the keyboard code is 84−32=52. Since each character in the set takes eight bytes, the letter T starts 8 × 52=416 bytes from the beginning of the character set RAM. Add these lines to change the letter T to a space:

```
400 REM MODIFY THE LETTER T
410 FOR X=0 TO 7
420 POKE CHARSET*256+416+X,0
430 NEXT X
500 REM SET CHARACTER BASE POINTER
510 POKE 756,CHARSET
```

Run it. Press the letter T after it says READY. Does it make a space? All of the other characters should be normal. List your program. Notice that everything is normal except for the T.

Challenge #18

Modify the ATARI character set to display this character in place of the upper case A.



Printing Characters to Graphics Screens

GRAPHICS modes 0, 1, and 2 are designed as character modes. However, sometimes there is a need to print in modes 3 through 8 as well. Let's first look at the two color modes 4, 6, and 8.

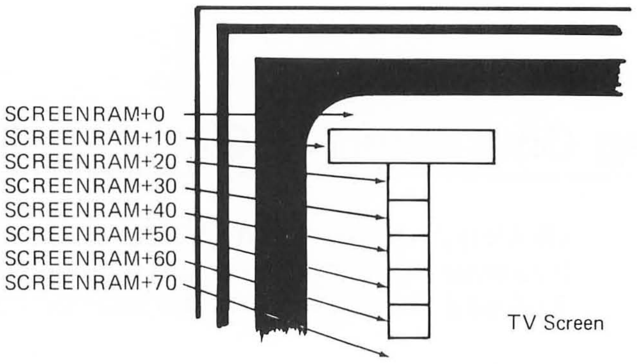
GRAPHICS Modes 4, 6, and 8

GRAPHICS modes 4, 6, and 8 are all two color modes. The pixels of any image displayed in these modes are either turned on or off, just as they are in GRAPHICS mode 0. Each pixel of a character requires one bit of screen memory. Characters are easy to display in the two color modes because characters in the character set ROM can be mapped directly into the screen RAM.

Let's try an example. Enter the following lines and run the program:

```
100 REM CHARACTERS IN MODES 4,6,AND 8
200 REM SET GRAPHICS MODE
210 GRAPHICS 4
300 REM FIND SCREEN RAM POSITION
310 SCREENRAM=PEEK(88)+PEEK(89)*256
400 REM PRINT THE LETTER T
420 FOR C=0 TO 7
430 POKE SCREENRAM+C*10,PEEK(224*256+416+C)
440 NEXT C
```

The letter T should be printed at the upper left hand corner of the screen. Line 430 requires an explanation. Since 10 bytes of screen memory are used for each horizontal mode line in GRAPHICS 4 (refer to Table 7-1), each line of the character T must be placed 10 bytes apart in screen memory.



The character definition data is read from the ATARI character generator which begins at address 224•256. We add 416 bytes to the generator address to locate the letter T just as we did before.

Change these lines to see the character T in GRAPHICS mode 6.

```
210 GRAPHICS 6

430 POKE SCREENRAM+C*20,PEEK(224*256+416+C)
```

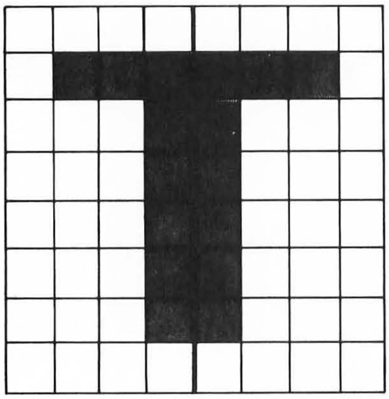
Notice that in mode 6, 20 bytes are required for each mode line.

GRAPHICS Modes 3, 5, and 7

We can also print to GRAPHICS modes 3, 5, and 7. These are four color modes. Because of this, each pixel displayed in these modes requires two bits of screen memory. The bit pairs and corresponding color registers are shown below:

Bit Pair	Color Register
00	Background
01	0
10	1
11	2

For characters to be displayed in GRAPHICS modes 3, 5, and 7, each line in the character matrix will require eight bit pairs or two bytes. For example, our letter T with a color of register 0 could be represented as shown. Notice that the bit pair 01 is used for each character dot that is to be displayed with the color of color register 0. Bit pair 00 is the background color.

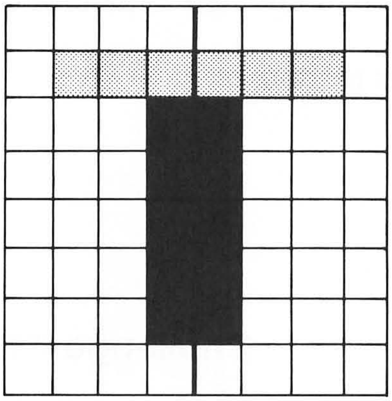
Character Image	Left Byte		Right Byte	
	00 00 00 00	= 0	00 00 00 00	= 0
	00 01 01 01	= 21	01 01 01 00	= 84
	00 00 00 01	= 1	01 00 00 00	= 64
	00 00 00 01	= 1	01 00 00 00	= 64
	00 00 00 01	= 1	01 00 00 00	= 64
	00 00 00 01	= 1	01 00 00 00	= 64
	00 00 00 01	= 1	01 00 00 00	= 64
	00 00 00 00	= 0	00 00 00 00	= 0

Sixteen bytes are required to represent the character. Let's poke the character data onto a GRAPHICS 5 screen to see the effect. Enter this program and run it.

```
100 REM CHARACTERS IN MODES 3,5,AND 7
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 5
220 SCREENRAM=PEEK(88)+PEEK(89)*256
300 REM DISPLAY LETTER T ON THE SCREEN
310 FOR X=0 TO 7
320 FOR I=0 TO 1
330 READ N:POKE SCREENRAM+X*20+I,N
340 NEXT I
350 NEXT X
390 DATA 0,0,21,84,1,64,1,64,1,64,1,64,1,64,0,0
```

In this routine, we didn't build or modify the character set. The character data for the letter T was placed into DATA statements. We could, however, place this character data in RAM just as we did earlier.

Now the exciting part comes. It's a simple matter to change some of the bit pairs to represent a different color register. Let's change the top of the letter T so that color register 2 is used instead of color register 0. The new diagram and values look like this.

Character Image	Left Byte	Right Byte
	00 00 00 00 = 0	00 00 00 00 = 0
	00 11 11 11 = 63	11 11 11 00 =252
	00 00 00 01 = 1	01 00 00 00 = 64
	00 00 00 01 = 1	01 00 00 00 = 64
	00 00 00 01 = 1	01 00 00 00 = 64
	00 00 00 01 = 1	01 00 00 00 = 64
	00 00 00 01 = 1	01 00 00 00 = 64
	00 00 00 00 = 0	00 00 00 00 = 0

Change the values in line 390.

```
390 DATA 0,0,63,252,1,64,1,64,1,64,1,64,1,64,0,
```

Now try it. Is the top of the T blue?

Multicolored characters can also be produced using ANTIC mode 4. You need to modify the display list for this mode.

Applications

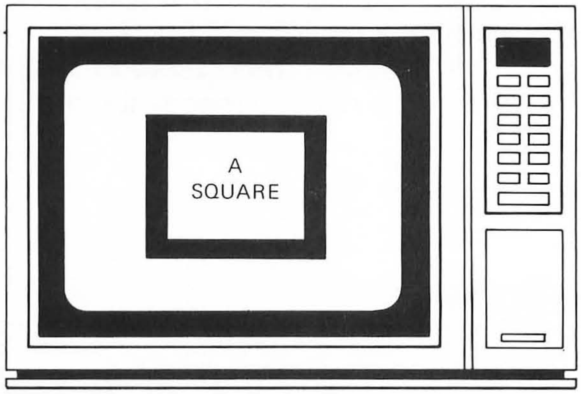
Perhaps you have a dozen ideas on how to use customized character sets. Here are some you may or may not have thought of:

1. Design and build a set of Greek characters or a specialized font of letters or numbers.
2. Design your own graphics characters and put them together to make a screen drawing.
3. Design multicolored characters—either letters and numbers or graphics designs.

4. Set up multiple character sets in RAM and switch back and forth as required.
5. Design a set of special characters used to draw electronics diagrams.
6. Use the character set for animation.

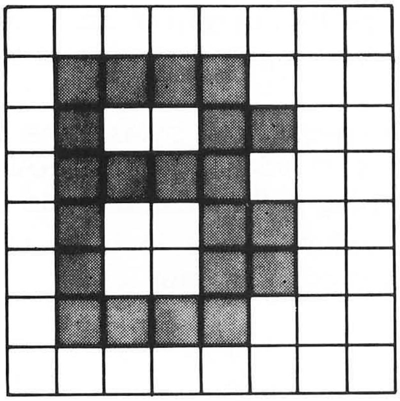
Challenge #19

Write a program to produce the following display in GRAPHICS mode 6.



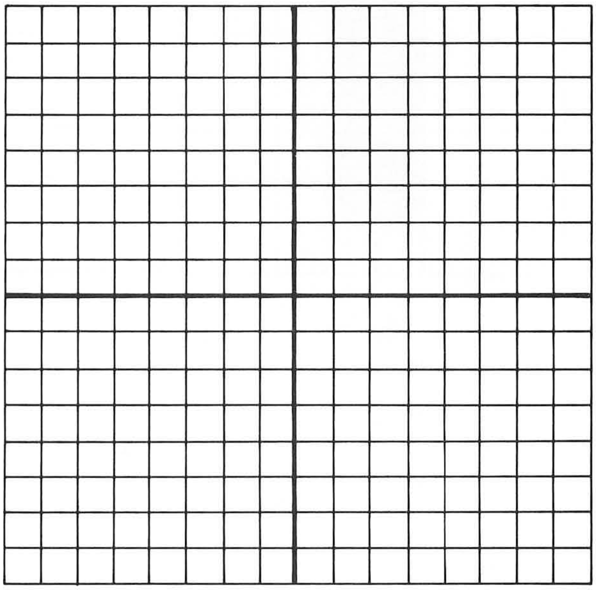
Chapter 10 Review

1. What decimal values could be used to represent each line of the character shown below?



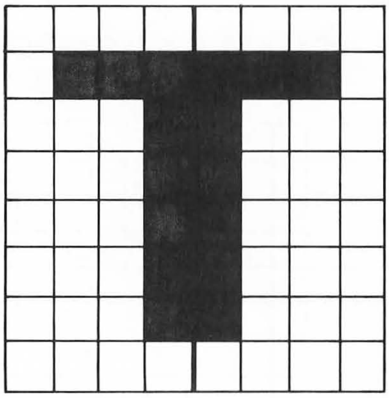
128	64	32	16	8	4	2	1	Decimal Value
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______
__	__	__	__	__	__	__	__	______

2. Draw a large letter T using four 8 × 8 characters as shown below.



3. How many bytes of memory are required for one single color character?

4. How many bytes of memory are required for a set of 128 single color characters?
5. What are the keyboard values for the following characters?
   a. 4
   b. S
   c. w
6. For a normal character set that begins at absolute memory address 56344, what is the beginning address for the letter A?
7. A GRAPHICS 6 mode line requires 20 bytes. If the left side of the first mode line begins at address SCREENRAM, what is the address of the beginning of the second mode line?
8. Calculate the decimal values of the 16 bytes required to represent the letter T using color register 2.

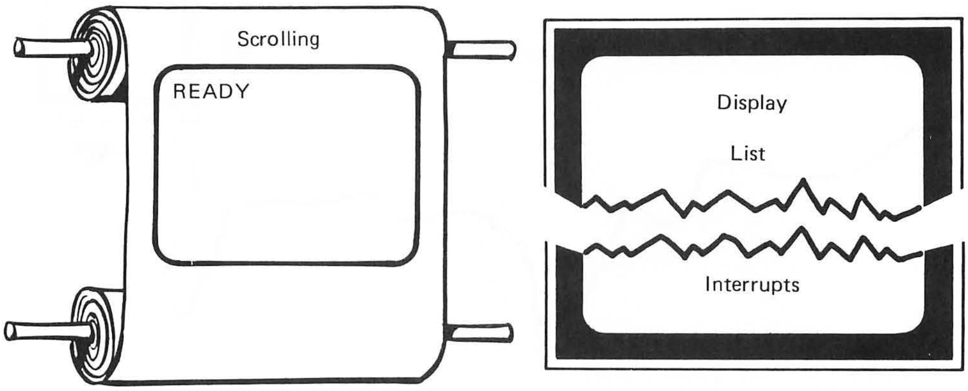


Left Byte	Right Byte

CHAPTER 11

The Display List

We studied display lists in Chapter 7 when we designed mixed mode screens. Display lists can be modified to produce some other interesting effects as well.

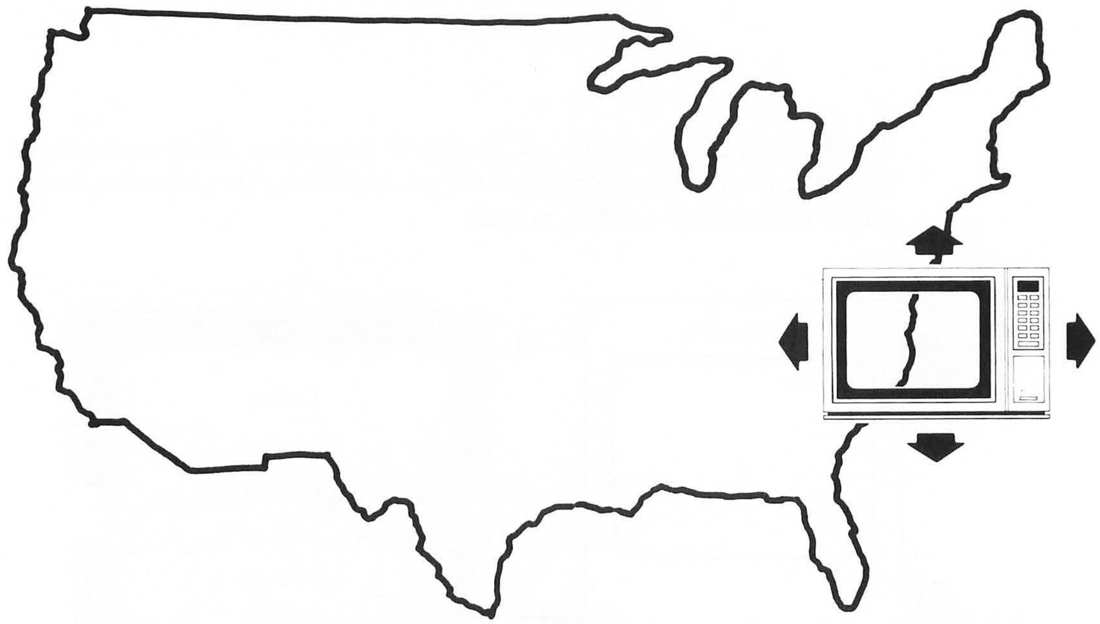


In this chapter you will learn to:

- Scroll images vertically and horizontally
- Use display list interrupts

Scrolling

Sometimes we wish to display more information than the screen will hold. For example, suppose that we wanted to display a detailed map of the United States. Neither size nor resolution of the screen would allow us to comfortably and accurately view it. We could, however, place the map data into screen memory and scroll through this data to display various parts of the map.



It's something like moving the TV screen around to display various parts of the map.

Vertical Scrolling

Those of us that use computers are familiar with vertical scrolling. Information is often scrolled vertically upward when listing a program. This information is scrolled a line at a time.

Generally speaking, data can be moved through the screen display RAM to produce scrolling. This can be a lengthy and complex process depending on the amount of data that needs to be moved. The ATARI computer, however, has a facility to simply change the screen display pointer and leave all of the screen data in place. This can allow for easier and quicker scrolling. Let's try an example to demonstrate what we mean. Enter the following program lines:

```
100 REM COARSE VERTICAL SCROLL IN MODE 0
200 REM FILL SCREENRAM AREA
210 SCREENRAM=PEEK(88)+PEEK(89)*256
220 FOR X=0 TO 959
230 POKE SCREENRAM+X,34
240 NEXT X
300 REM INCREMENT SCREENRAM POINTER
310 DLIST=PEEK(560)+PEEK(561)*256
320 FOR X=1 TO 20
330 SCREENRAM=SCREENRAM+40
340 HI=INT(SCREENRAM/256)
350 LO=SCREENRAM-HI*256
355 REM POKE SCREEN ADDRESS INTO DISPLAY LIST
360 POKE DLIST+4,LO:POKE DLIST+5,HI
370 FOR WAIT=1 TO 100:NEXT WAIT
380 NEXT X
```

Run it. The screen display data is scrolled off the screen.

Recall from Chapter 7 that the display list for a GRAPHICS 0 screen looks as follows:

```
112 )
112 }→ Blank eight lines (three times)
112 )
66  → LMS Instruction
64  }→ Address of screen display RAM
156 }  (These values will be different if you don't have 48K of
2   )  RAM in your computer)
2   |
2   |
2   }→ Mode bytes for GRAPHICS 0
2   |
2   |
2   )
```

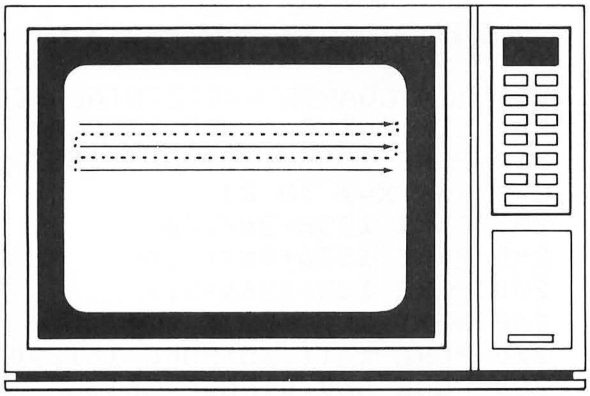
```
2   )
2   |
2   |
2   |
2   |--> Mode bytes for GRAPHICS 0
2   |
2   |
2   |
2   |
2   |
2   |
2   |
2   |
2   |
2   |
2   )
65  ---> Jump instruction
32  \__, Address of display list RAM
156 /    (These values will be different if you don't have 48K
         of RAM in your computer)
```

The LMS instruction in the display list points to the screen memory address. By changing the value of this screen memory address, we are simply moving the upper left hand corner of the screen to a new position in the screen memory. Since GRAPHICS mode 0 screens contain 40 bytes per line, moving the screen memory pointer 40 bytes at a time will cause the screen to scroll one line at a time. This is actually a *coarse scroll.* The screen appears to move in increments of one mode line or eight scan lines.

Horizontal Scrolling

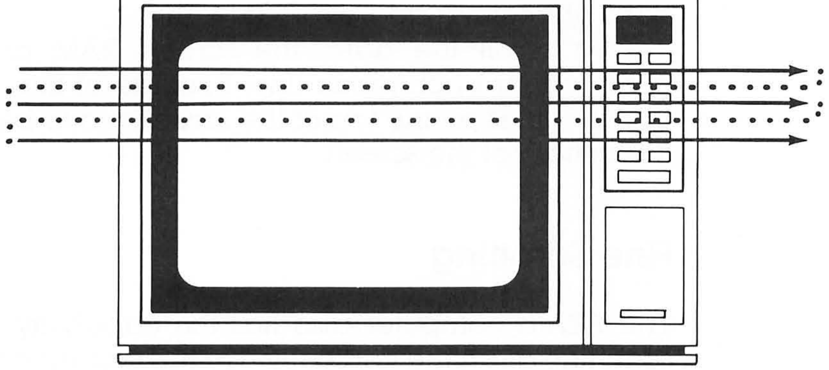
Horizontal scrolling is not as simple as vertical scrolling. Let's take a closer look at the screen memory arrangement.

Screen memory is arranged sequentially from the left side of the screen to the right. The byte representing the beginning of the second mode line would follow the byte representing the right end of the first mode line. An attempt to move the screen memory pointer one byte forward would result in the character at the left side of the second line popping into the position at the right side of the first line. The following diagram shows the sequence of screen data in memory:



The address of the screen position increases from left to right and from top to bottom.

Because of this screen memory arrangement, horizontal scrolling cannot be accomplished like vertical scrolling. The problem can be overcome by making each horizontal line bigger than the screen. This means that additional screen memory will be required. The following diagram shows what the screen memory arrangement will need to look like:



GRAPHICS modes, however, do not normally have display lists which allow for added memory space in each mode line. This problem can be solved by writing a custom display list. Each mode line will require a display list instruction to set the screen RAM address for the line. It's not as difficult as it might sound.

The following program will produce coarse horizontal scroll. Enter the lines and try it:

```
100 REM COARSE HORIZONTAL SCROLL IN MODE 0
200 REM BUILD A NEW DISPLAY LIST AT PAGE SIX
210 POKE 1536,112:POKE 1537,112:POKE 1538,112
220 FOR X=1 TO 24
230 POKE 1536+3*X,66
240 POKE 1536+3*X+1,0
250 POKE 1536+3*X+2,X
260 NEXT X
270 POKE 1611,65:POKE 1612,0:POKE 1613,6
280 POKE 560,0:POKE 561,6
300 REM SCROLL RAM DATA
310 FOR X=0 TO 215
320 FOR Y=1 TO 24
330 POKE 1536+3*Y+1,X
340 NEXT Y
350 NEXT X
360 GOTO 300
```

The display list we built is a little different than normal. Instead of a single mode byte for each mode line, we have used an LMS byte followed by the screen RAM address for each mode line. The display list has been placed in page 6 of RAM which starts at decimal address 1536.

To scroll the data, the screen RAM address must be changed for each mode line. Because of this, you may notice that the lines on the top of the screen move before the lines at the bottom of the screen.

Fine Scrolling

The ATARI computer also has the capability of scrolling one scan line at a time vertically. Try this example to see the effect:

```
100 REM VERTICAL FINE SCROLL
200 REM ENABLE VERTICAL FINE SCROLL BIT IN DISPLAY LIST
210 DLIST=PEEK(560)+PEEK(561)*256
220 FOR X=10 TO 20:POKE DLIST+X,34:NEXT X
300 REM FILL SCREENRAM WITH DATA
310 SCREENRAM=PEEK(88)+PEEK(89)*256
320 FOR X=0 TO 959:POKE SCREENRAM+X,34:NEXT X
400 REM EXECUTE FINE SCROLL
410 FOR Y=0 TO 7
420 POKE 54277,Y
425 FOR WAIT=1 TO 50:NEXT WAIT
430 NEXT Y
440 GOTO 400
```

First, the vertical fine scroll bit (bit 5) is set for each mode line that is to be scrolled. The number of scan lines of vertical movement is then POKED into the vertical fine scroll register. No more than 16 scan lines can be fine scrolled. To move images over the entire height of the screen requires a combination of coarse and fine scrolling.

You may notice that the screen seems to blink once in a while. This is due to the changing of values in the fine scroll register. Normally, this register should only be changed while the electron beam is moving from the bottom of the screen back to the top. This is called the *vertical blank*. Access to vertical blank interrupts requires machine language routines. Good quality scrolling will require assembly language programming.

Display List Interrupts

We have saved one of the more exciting features for last. The use of display list interrupts is normally not available to the BASIC programmer but we can nonetheless do a couple of examples.

Let's find out what a display list interrupt can do. Try the following program:

```
100 REM DISPLAY LIST INTERRUPT DEMONSTRATION
200 REM SET DLI IN DISPLAY LIST
210 DLIST=PEEK(560)+PEEK(561)*256
```

```
220 POKE DLIST+10,130
300 REM PUT DLI SERVICE ROUTINE IN PLACE
310 FOR X=0 TO 10
320 READ N:POKE 1536+X,N
330 NEXT X
360 DATA 72,169,22,141,10,212,141,24,208,104,64
500 REM ENABLE DLI
510 POKE 512,0:POKE 513,6
520 POKE 54286,192
```

Is the bottom part of the screen orange? It should be.

The program works something like this. Any mode byte in the display list can be set to trigger an interrupt service routine. Bit 7 of the mode byte sets the display list interrupt. This is equivalent to adding 128 to the normal mode byte for the GRAPHICS mode you are using.

When this interrupt is encountered in the normal execution of the display list, the computer checks to see if the interrupt is enabled by looking at address 54286. If the value is 192, it is enabled. The computer then checks addresses 512 and 513 to find at which address the interrupt service routine is located. In this case, the routine is located at page 6 or decimal address 1536.

The interrupt service routine is written in machine language. In this case, the routine changes the color value in color register 2. When the mode line with the display list interrupt bit set is encountered, the background color is changed to orange. It is reset to blue during the vertical blank process and, thus, the top part of the screen remains blue. Let's take a brief look at the assembler code for the interrupt service routine.

Assembler Code	Decimal Codes
PHA	72
LDA COLOR	169,22
STA REG2	141,10,212
STA WSYNC	141,24,208
PLA	104
RTI	64

An explanation at this point is not meaningful unless you are familiar with assembly language. Very simply, the assembler code is used to store a color (22 in this case) into color register 2.

Let's make a couple of changes and add another interrupt to a different mode line. We will also add another interrupt service routine. Change and add these lines:

```
230 POKE DLIST+17,130

310 FOR X=0 TO 15

400 REM 2ND DLI SERVICE ROUTINE
410 FOR X=0 TO 15
420 READ N:POKE 1556+X,N
430 NEXT X
460 DATA 72,169,66,141,10,212,141,24,208,169,
0,141,0,2,104,64
```

Run the program. We have added yet another color to the screen. In fact, we could make each mode line on the GRAPHICS 0 screen a different color. Again, machine language is required. Perhaps now you have the incentive to learn assembly language programming.

The Finishing Touch

Since you've reached this point, I suppose you are looking for some words of wisdom. I'll make no promises about the words being wise; however, here are some thoughts for you to ponder.

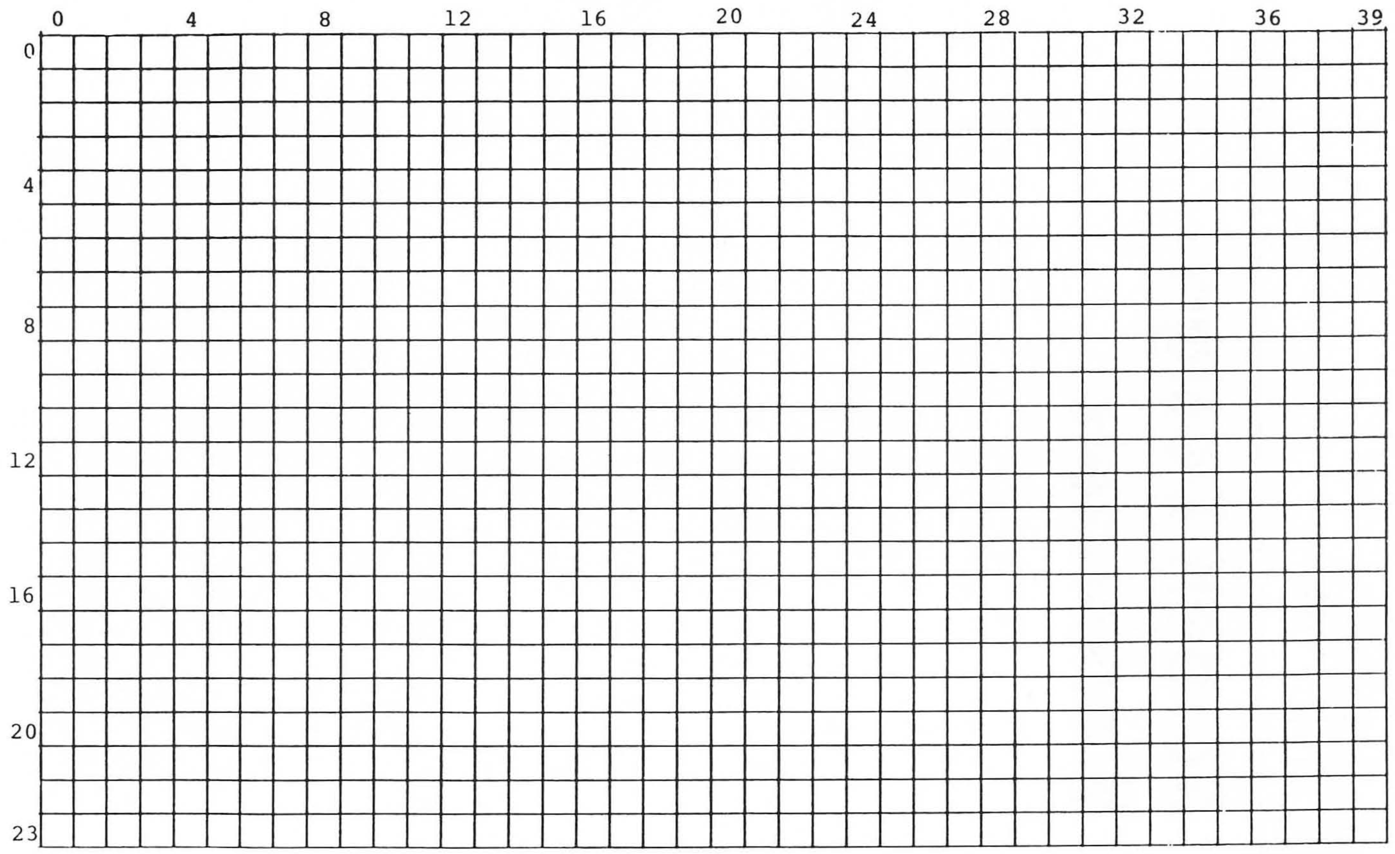
One of the greatest thrills of using a computer is to have control over the process and the medium. You can not only create a graphic screen image but you can change it, manipulate it, and recreate it. You have the power to give it life, to control its movement and appearance.

This book has been concerned with using the tools of ATARI graphics. You alone have the opportunity to design and create animated graphic images never known to another person. Why does this opportunity belong only to you? Because the images you create are the "Designs from Your Mind."

APPENDIX A

Screen Design Worksheets

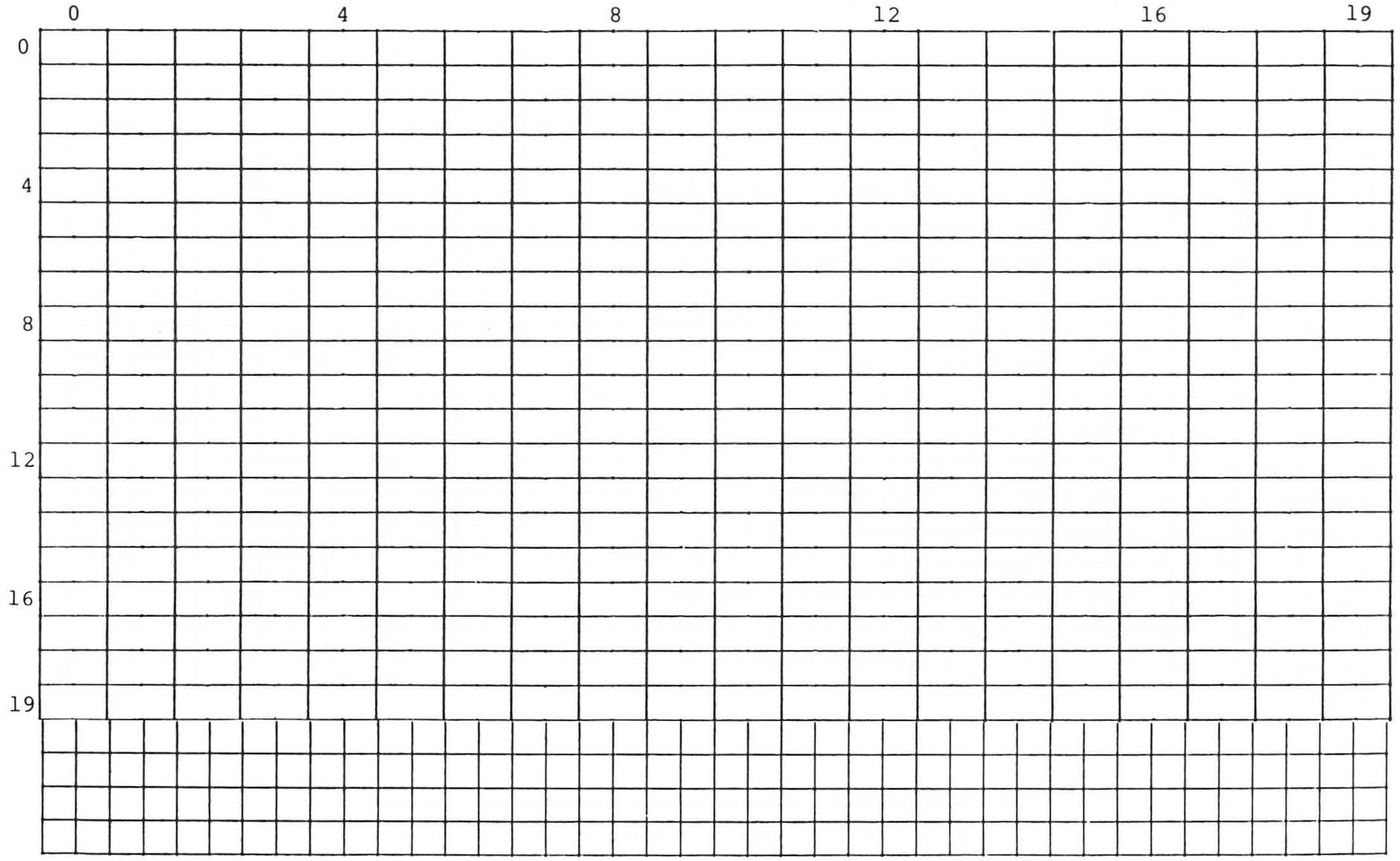
Graphics Mode 0



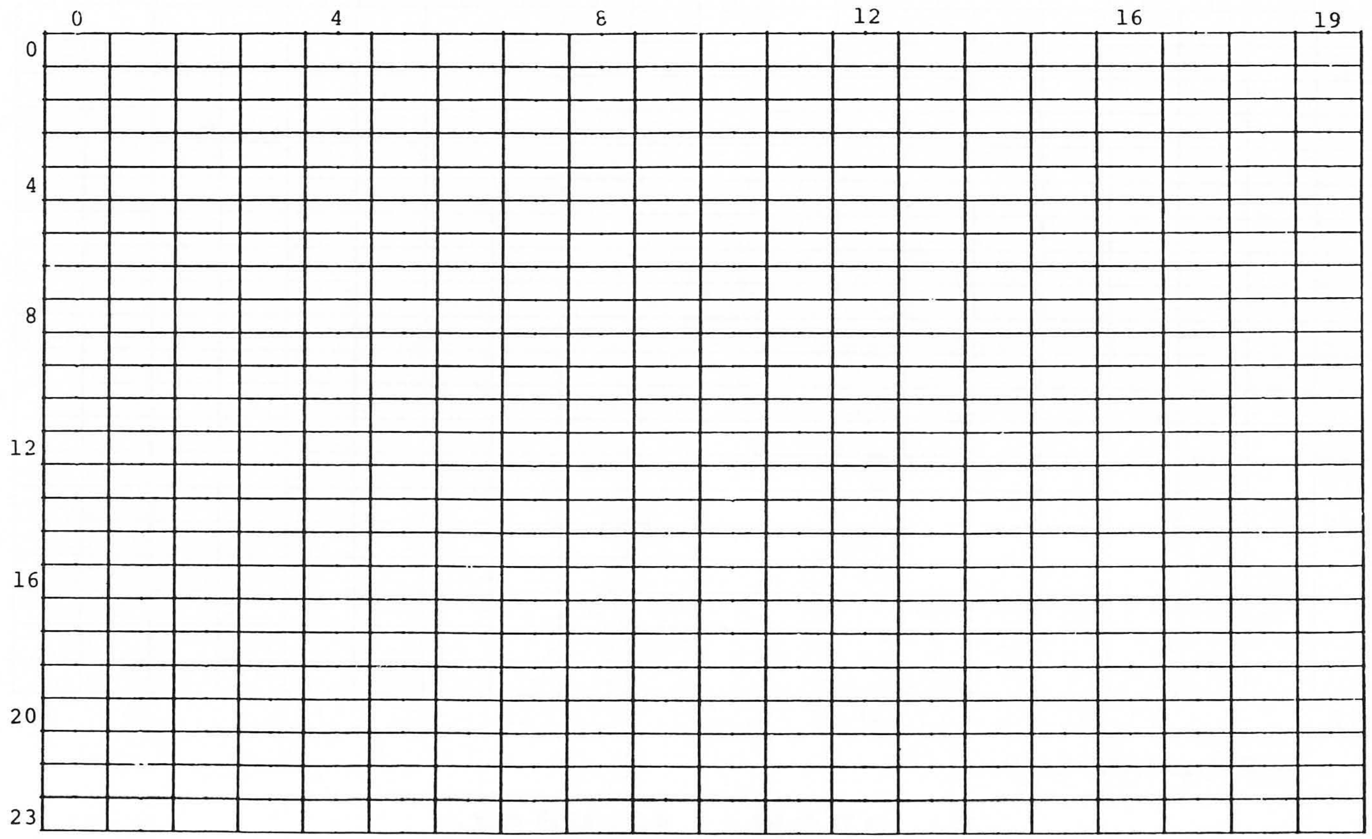
Notes: ______________________________

Graphics Mode 1

with Text Window

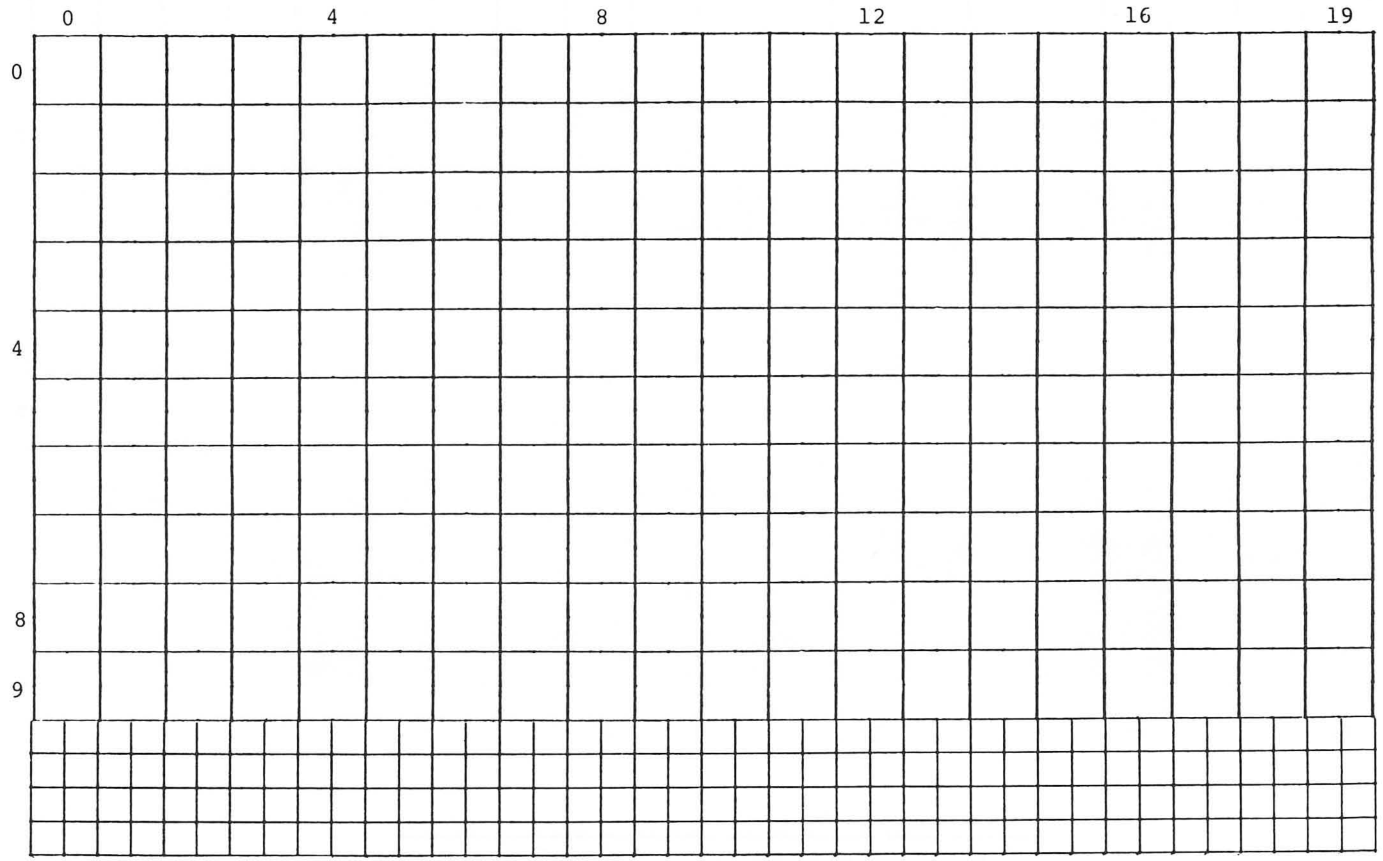


Graphics Mode 1

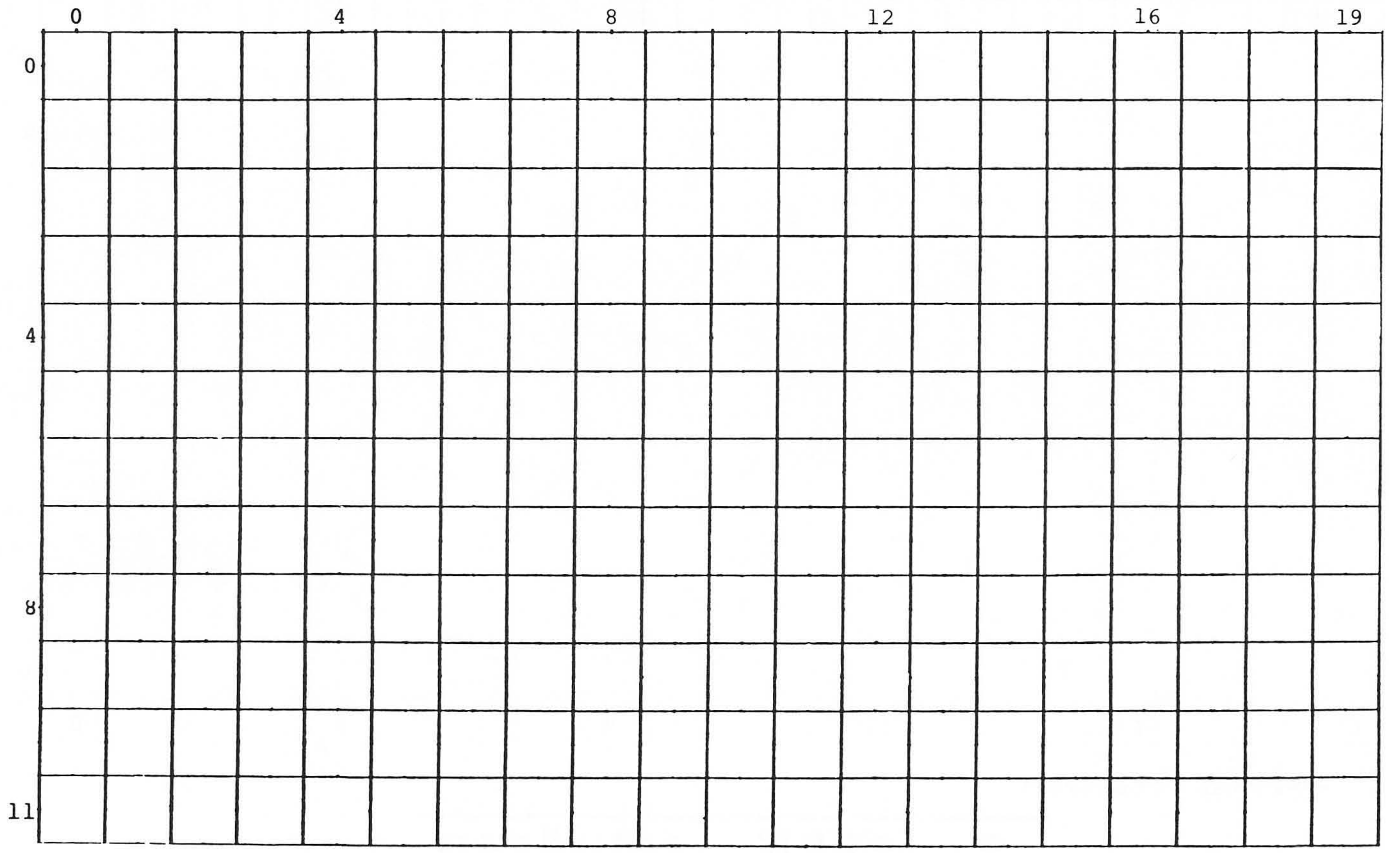


Graphics Mode 2

with Text Window

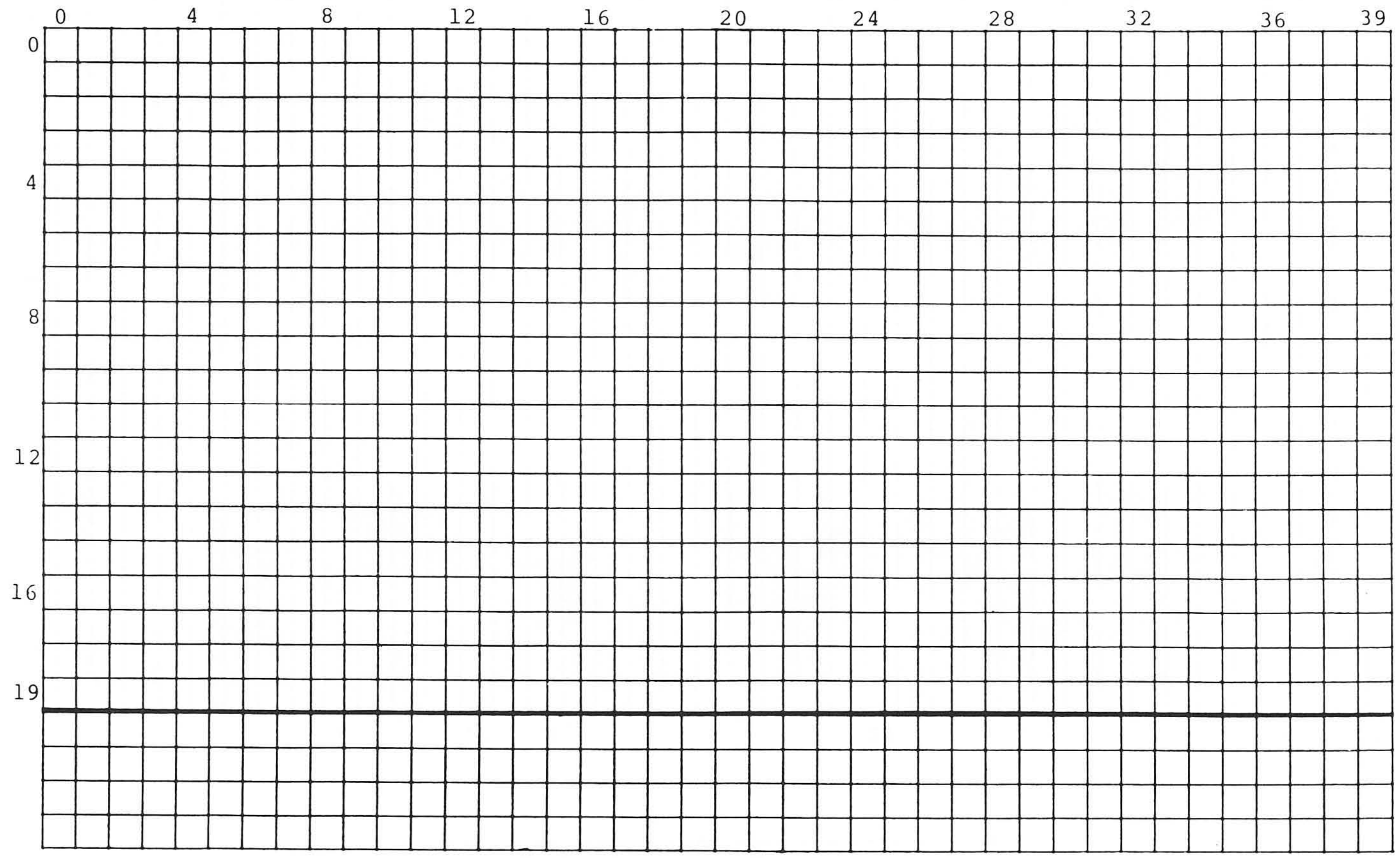


Graphics Mode 2

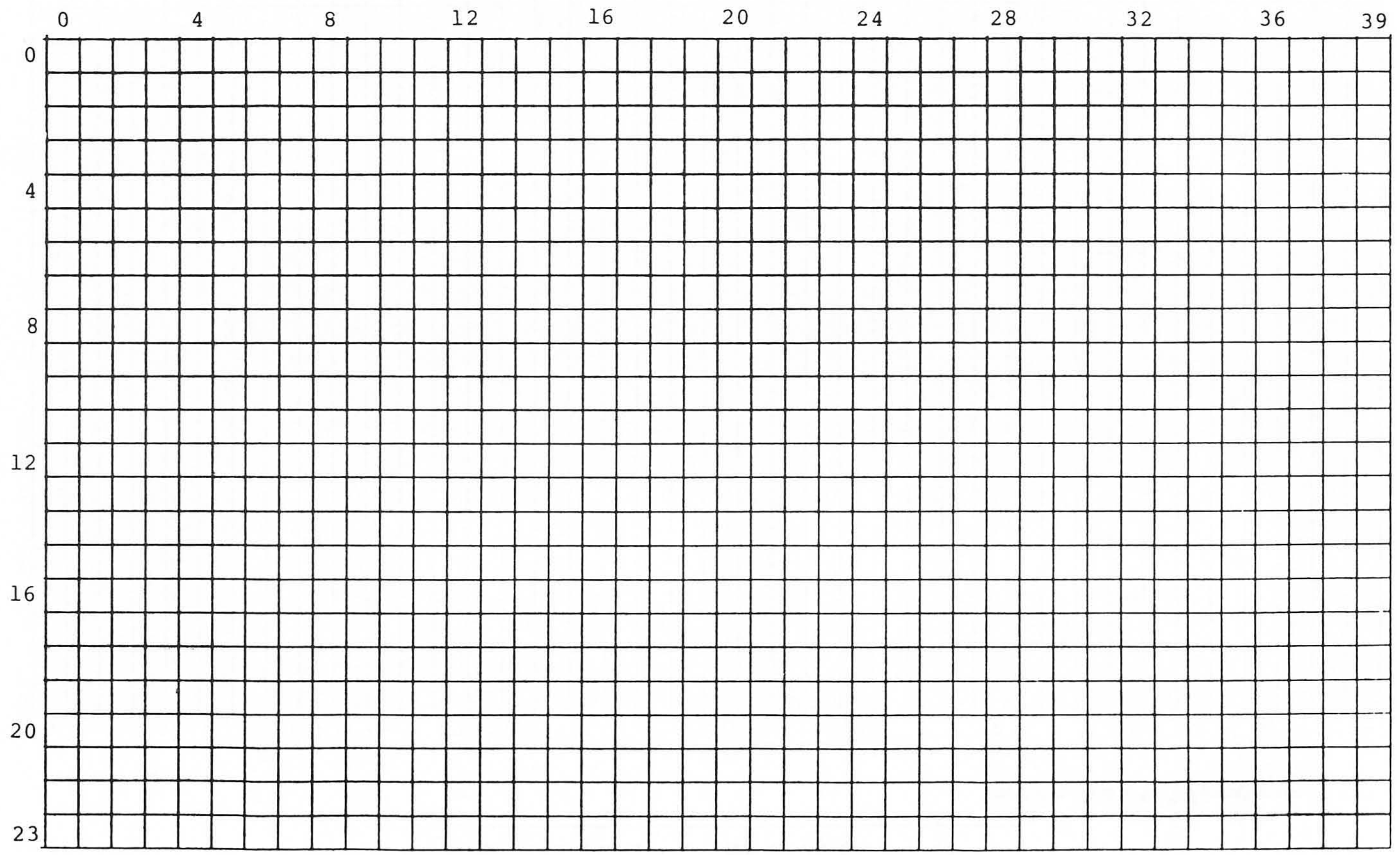


Graphics Mode 3

with Text Window

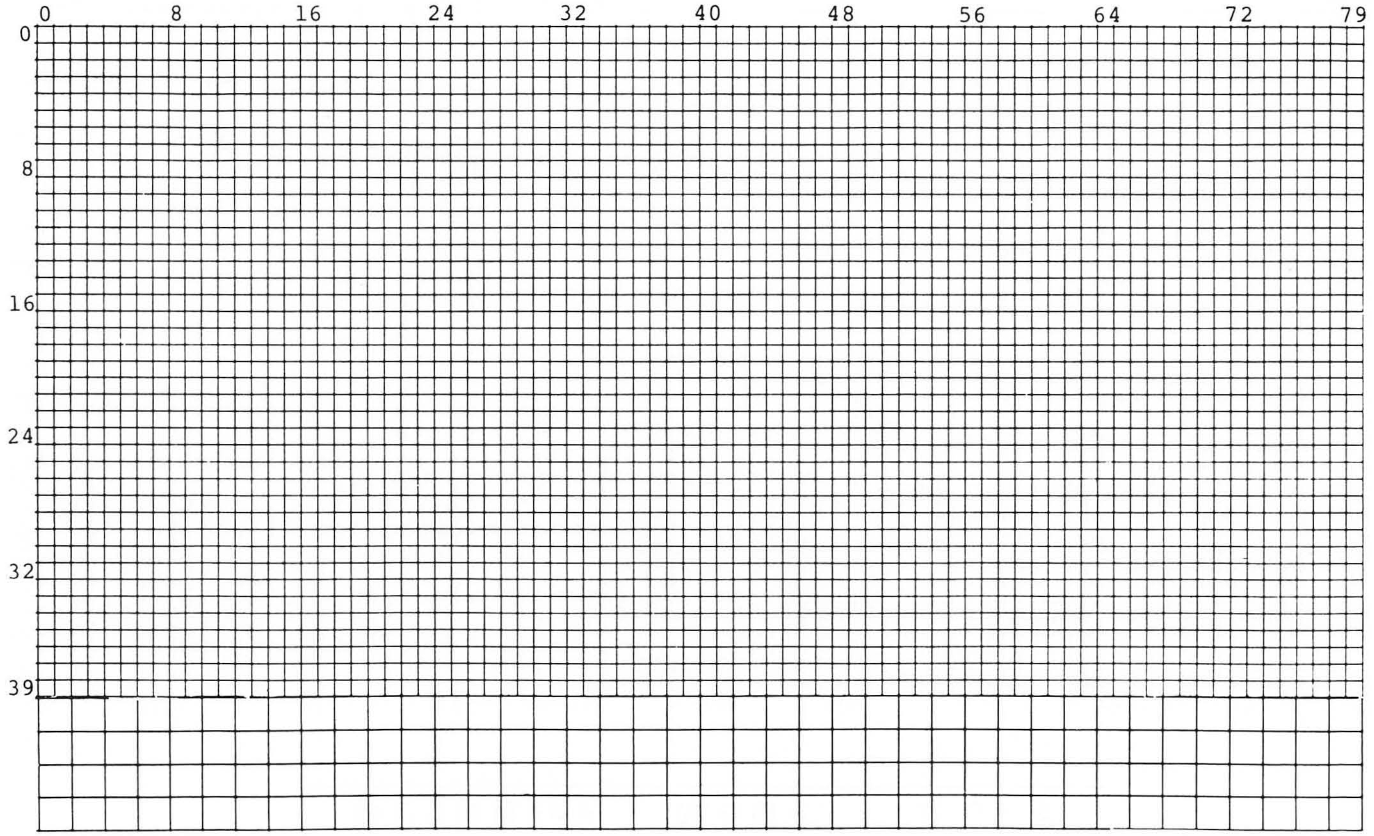


Graphics Mode 3

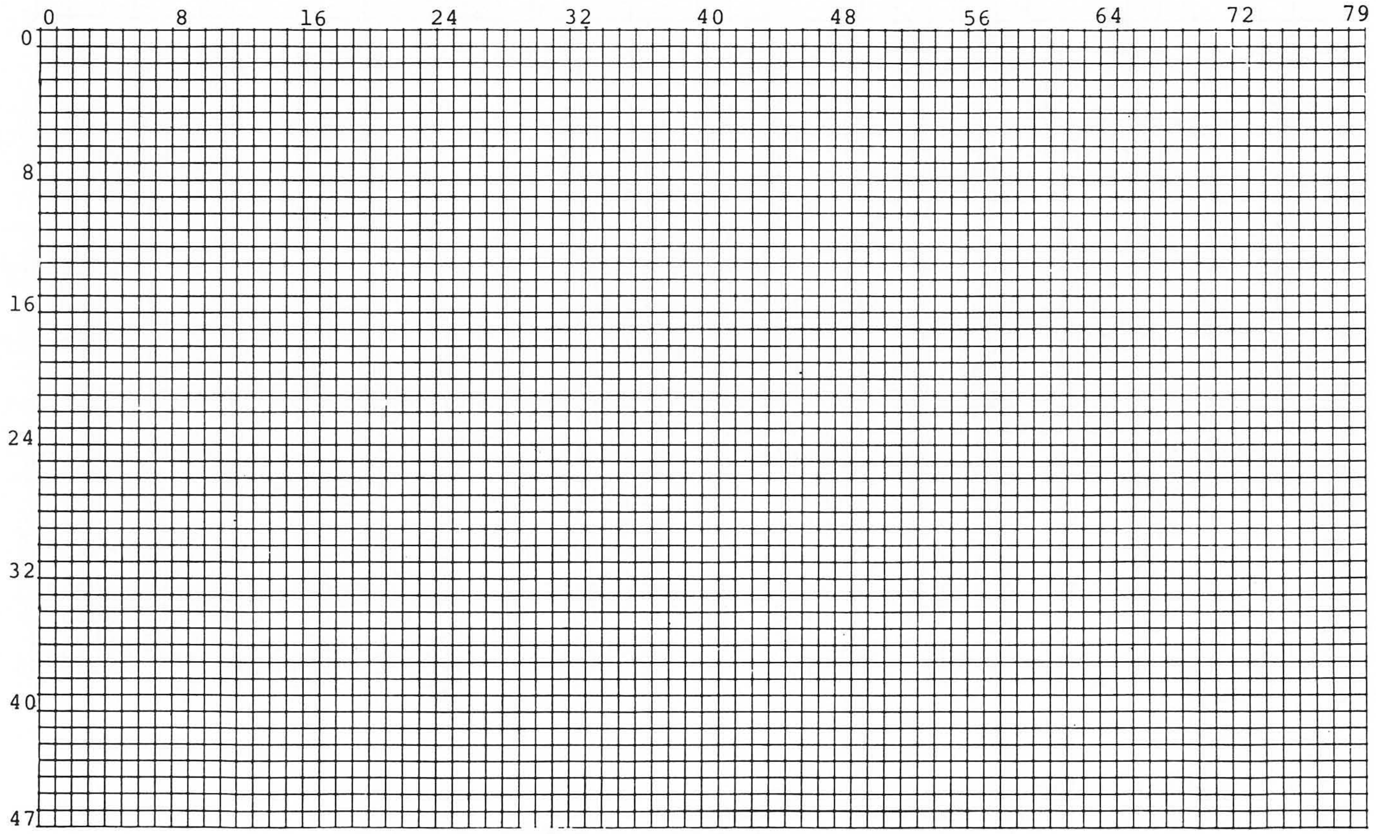


Graphics Mode 4 or 5

with Text Window

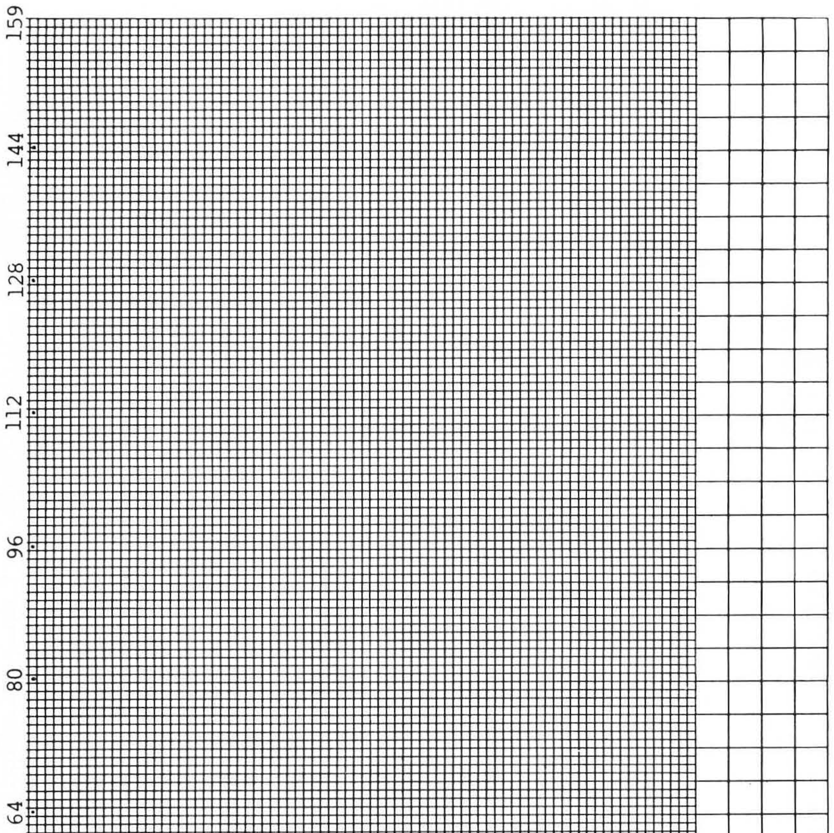


Graphics Mode 4 or 5

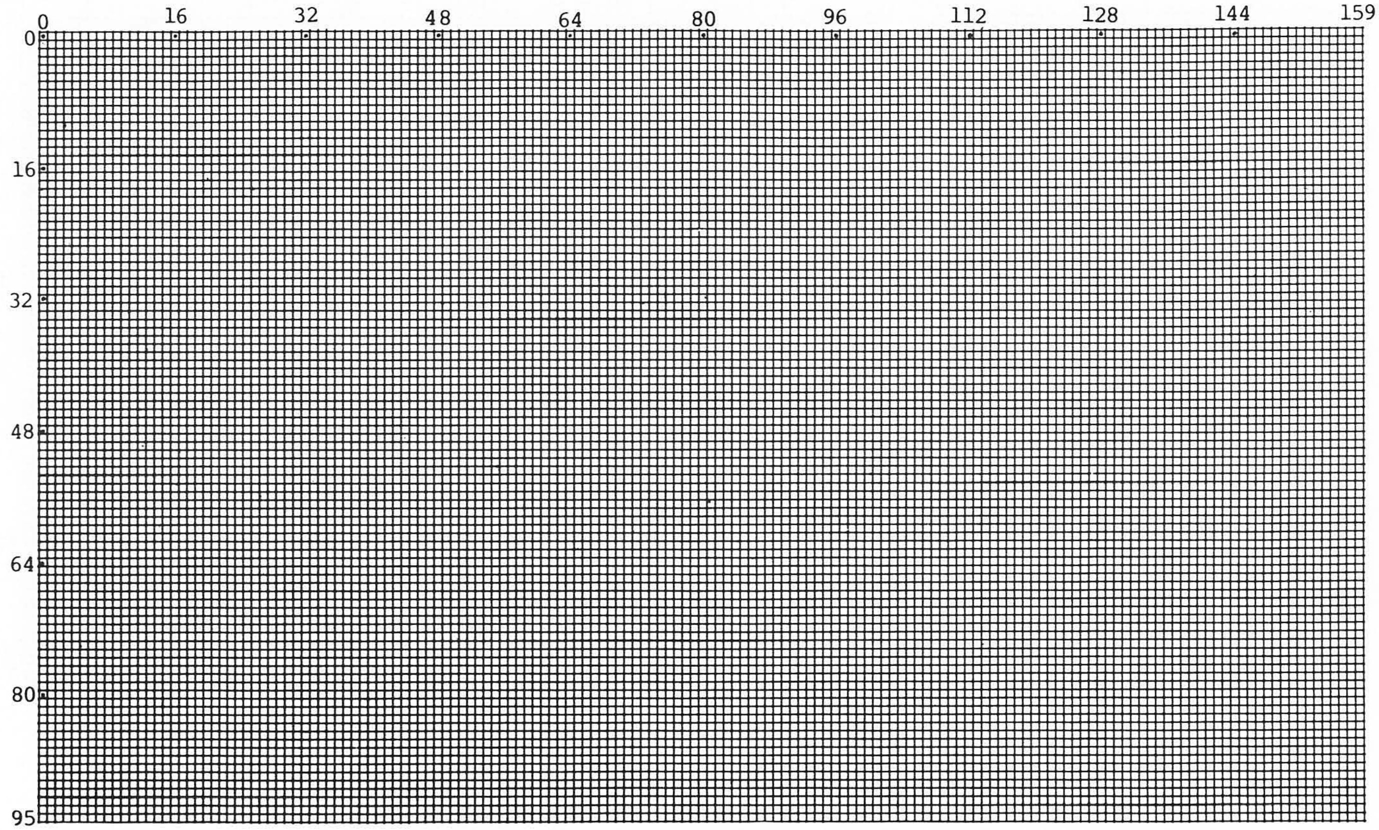


Graphics Mode 6 or 7

with Text Window

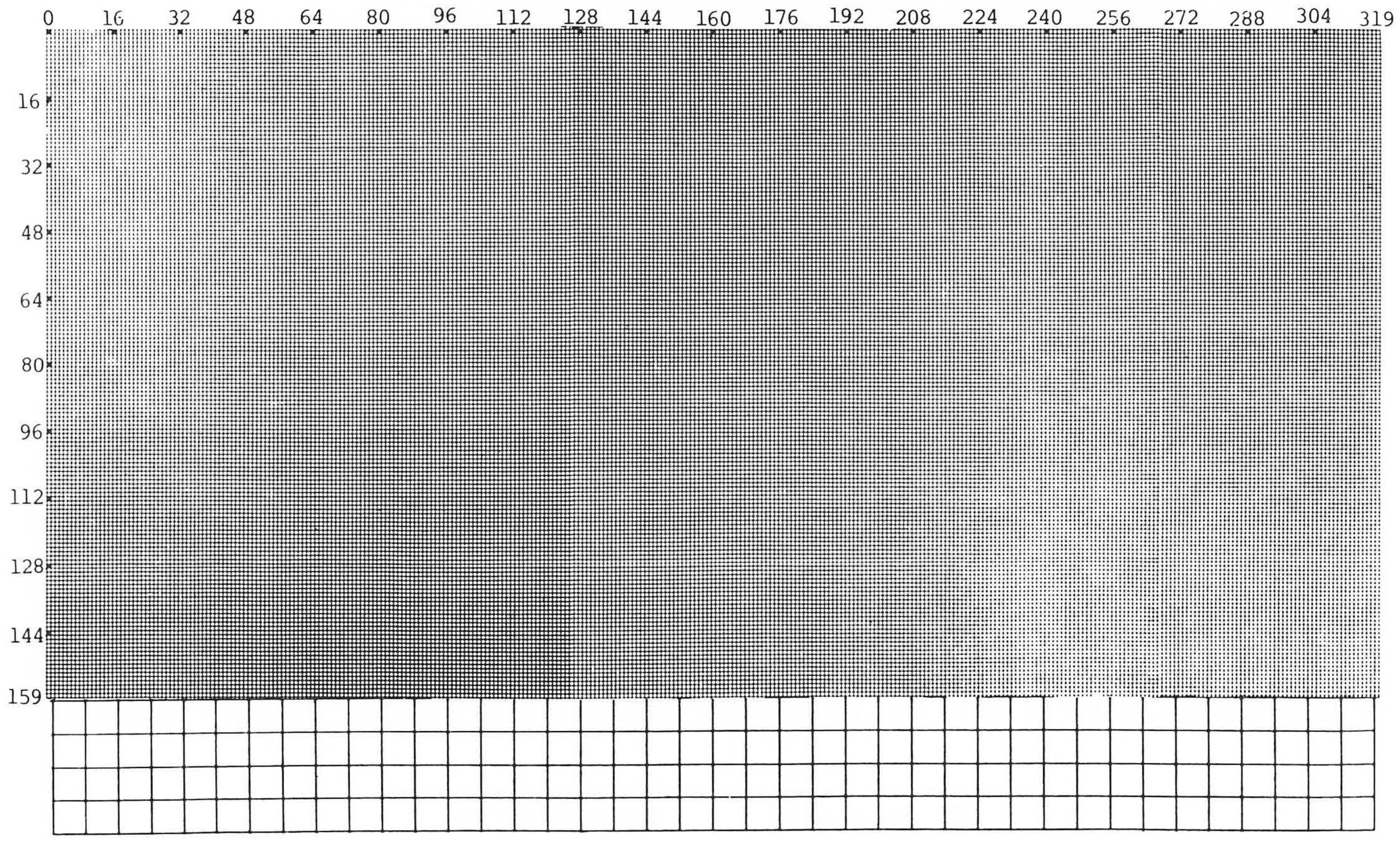


Graphics Mode 6 or 7

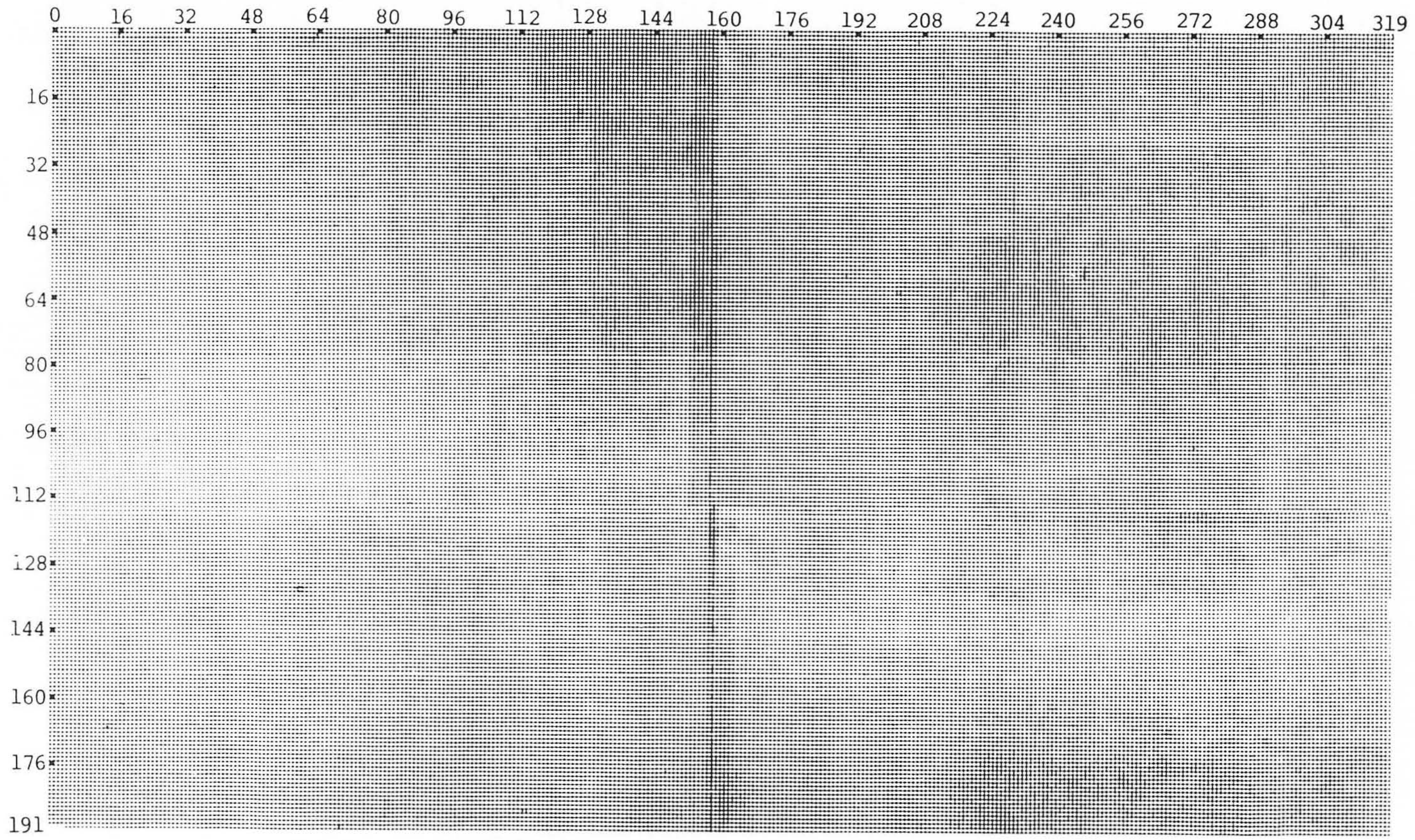


Graphics Mode 8

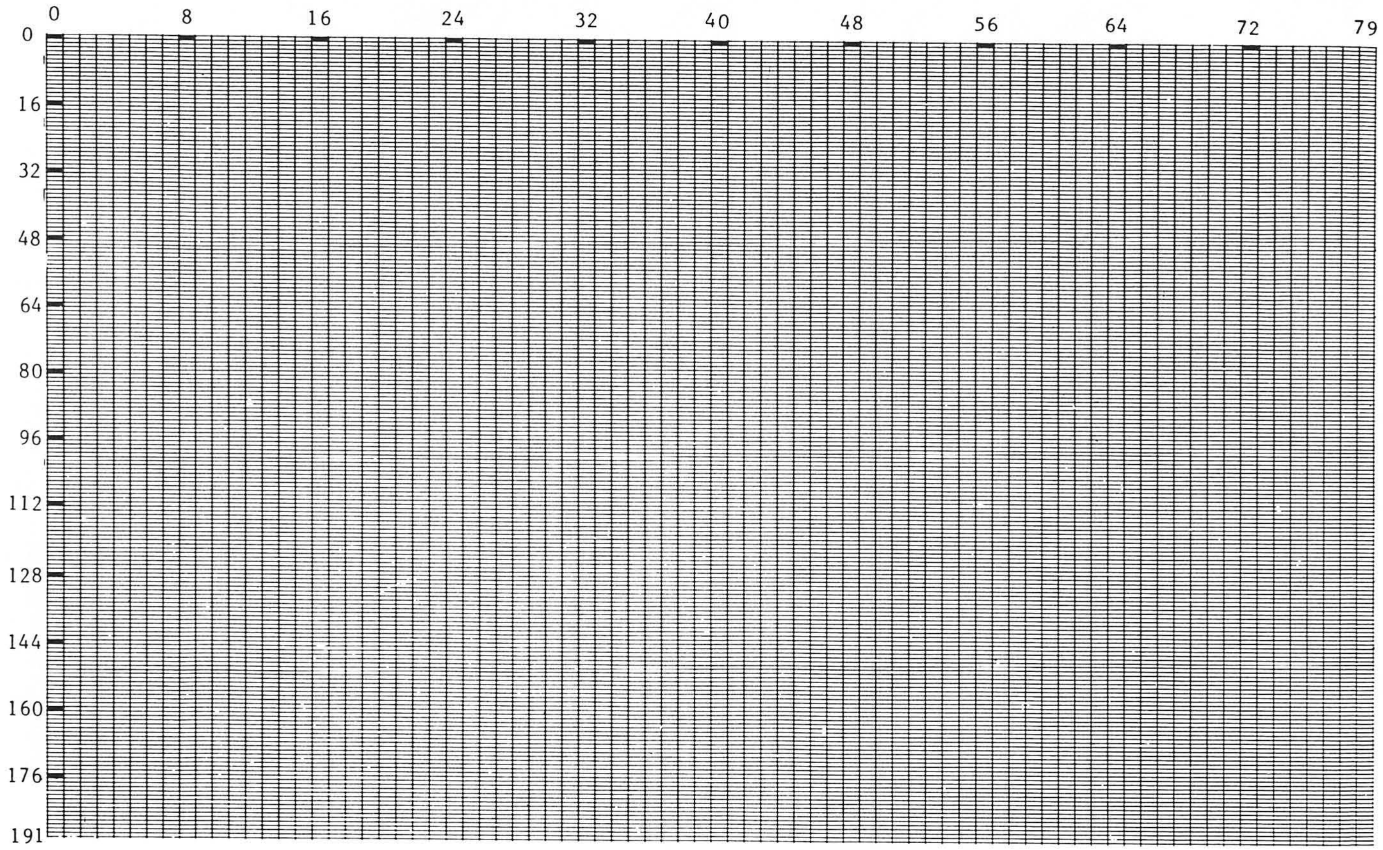
with Text Window



Graphics Mode 8

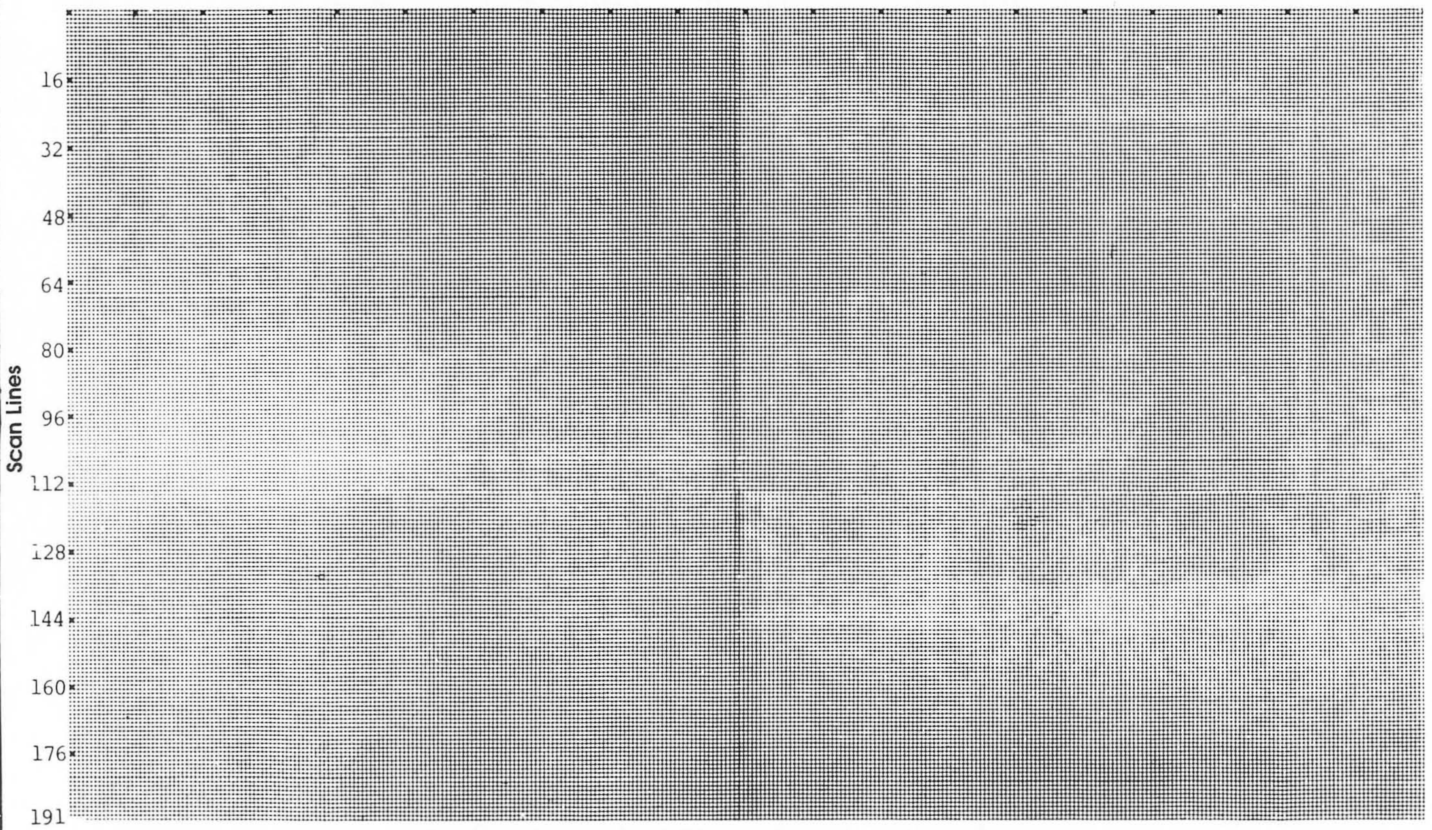


Graphics Mode 9, 10, or 11



GTIA

Display List Worksheet



Character Design Worksheet

128	64	32	16	8	4	2	1	Decimal Value

128	64	32	16	8	4	2	1	Decimal Value

128	64	32	16	8	4	2	1	Decimal Value

APPENDIX B

Solutions to Challenges

1.
```
SETCOLOR 2,2,6     (Orange background)
SETCOLOR 4,8,6     (Blue Border)
SETCOLOR 1,2,0     (Dark Words)
```

2.
```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 10,5
40 DRAWTO 19,5
50 DRAWTO 19,14
60 DRAWTO 10,14
70 DRAWTO 10,5
```

3.
```
10 GRAPHICS 3
20 COLOR 1
30 PLOT 17,7
40 DRAWTO 22,7
50 DRAWTO 22,12
60 DRAWTO 17,12
70 DRAWTO 17,7
```

4.
```
100 REM HOUSE IN GRAPHICS 7
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 SETCOLOR 0,4,4
```

```
230 SETCOLOR 1,8,4
300 REM DRAW THE TOP OF THE HOUSE
310 COLOR 1
330 PLOT 79,30
340 DRAWTO 99,40
350 DRAWTO 59,40
360 DRAWTO 79,30
400 REM DRAW THE BOTTOM OF THE HOUSE
410 COLOR 2
420 PLOT 64,41
430 DRAWTO 94,41
440 DRAWTO 94,61
450 DRAWTO 64,61
460 DRAWTO 64,41
500 END
```

5.

```
100 REM COLOR CONTRAST DISPLAY
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
220 SETCOLOR 0,2,8
230 SETCOLOR 1,12,8
240 SETCOLOR 2,6,8
250 SETCOLOR 4,0,0
300 REM DRAW 3 BLOCK IN COLORS 1,2,3
310 FOR C=0 TO 2
320 COLOR C+1
330 REM DRAW 7 HORIZONTAL LINES
340 FOR X=1 TO 7
350 PLOT 2,C*7+X
360 DRAWTO 38,C*7+X
370 NEXT X
380 NEXT C
390 GOTO 390
400 END
```

(*Note:* The luminance of the colors may vary.)

6. (1)

```
100 REM COLOR CONTRAST DISPLAY
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
220 SETCOLOR 0,4,4
230 SETCOLOR 1,6,4
240 SETCOLOR 2,4,4
```

```
250 SETCOLOR 4,0,0
300 REM DRAW 3 BLOCK IN COLORS 1,2,3
310 FOR C=0 TO 2
320 COLOR C+1
330 REM DRAW 7 HORIZONTAL LINES
340 FOR X=1 TO 7
350 PLOT 2,C*7+X
360 DRAWTO 38,C*7+X
370 NEXT X
380 NEXT C
390 GOTO 390
400 END
```

(2)

```
100 REM COLOR CONTRAST DISPLAY
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 3+16
220 SETCOLOR 0,6,4
230 SETCOLOR 1,6,4
240 SETCOLOR 2,6,4
250 SETCOLOR 4,4,4
300 REM DRAW 3 BLOCK IN COLORS 1,2,3
310 FOR C=0 TO 2
320 COLOR C+1
330 REM DRAW 7 HORIZONTAL LINES
340 FOR X=1 TO 7
350 PLOT 2,C*7+X
360 DRAWTO 38,C*7+X
370 NEXT X
380 NEXT C
390 GOTO 390
400 END
```

7.

```
100 REM SIDEWALK-1 POINT PERSPECTIVE
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 COLOR 1
300 REM READ DATA AND DRAW LINES
310 READ LINES
320 FOR L=1 TO LINES
330 READ P1,P2,D1,D2
340 PLOT P1,P2:DRAWTO D1,D2
```

```
350 NEXT L
360 DATA 7
370 DATA 10,10,39,69,10,10,119,69,39,69,119,69
380 DATA 28,47,80,47,22,32,50,32,17,23,33,23
390 DATA 14,16,20,16
800 END
```

8.

```
100 REM SIDEWALK-1 POINT PERSPECTIVE
200 REM SET UP GRAPHICS SCREEN
210 GRAPHICS 7
220 COLOR 1
230 SETCOLOR 0,0,8
300 REM READ DATA AND DRAW LINES
310 READ LINES
320 FOR L=1 TO LINES
330 READ P1,P2,D1,D2
340 PLOT P1,P2:DRAWTO D1,D2
350 NEXT L
360 DATA 7
370 DATA 60,20,20,25,20,25,20,45,20,45,60,55
380 DATA 60,55,110,43,110,43,110,27,110,27,60,20
390 DATA 60,20,60,55
400 REM FILL RIGHT SIDE WITH COLOR
410 POKE 765,1
420 PLOT 60,20
430 POSITION 60,55
440 XIO 18,#6,0,0,"S:"
800 END
```

9.

```
10 REM VERTICAL LINES IN GRAPHICS 0
20 GRAPHICS 0
30 FOR X=1 TO 760
40 PRINT CHR$(22);
50 NEXT X
```

10.

```
10 REM CHALLENGE #10
20 GRAPHICS 2+16
40 POSITION 7,1:PRINT #6;"COLORS"
50 POSITION 0,4
60 PRINT #6;"  BLUE"
70 PRINT #6;"  gREEN"
80 PRINT #6;"  rED"
100 GOTO 100
```

11.
```
7000 REM A CLOUD
7010 FOR T=1 TO 100
7015 COLOR 2
7020 PLOT RND(0)*15+25,RND(0)*10+15
7030 PLOT RND(0)*25+20,RND(0)*20+10
7050 NEXT T
```

12.
```
10 REM CHALLENGE # 12
15 GRAPHICS 5+16
20 DLIST=PEEK(560)+PEEK(561)*256
30 REM 3*8 BLANK LINES
40 POKE DLIST,112
50 POKE DLIST+1,112
60 POKE DLIST+2,112
70 REM SET LMS OPTION
80 POKE DLIST+3,70
90 REM SCREEN MEMORY LOCATION
100 POKE DLIST+4,PEEK(88)
110 POKE DLIST+5,PEEK(89)
120 REM ANOTHER MODE 1 LINE
130 POKE DLIST+6,6
140 REM MODE 5 LINES
150 FOR X=7 TO 44:POKE DLIST+X,10:NEXT X
160 REM MODE 2 LINES
170 FOR X=45 TO 46:POKE DLIST+X,7:NEXT X
180 REM SET JUMP INSTRUCTION
190 POKE DLIST+47,65
200 REM BEGINNING OF DISPLAY LIST
210 POKE DLIST+48,PEEK(560)
220 POKE DLIST+49,PEEK(561)
300 REM WRITE TO SCREEN
310 SCREENMEM=PEEK(88)+PEEK(89)*256
320 POKE 87,1
325 POSITION 0,0
330 PRINT #6;"   A COLORED BLOCK"
335 REM DRAW COLORED BLOCK
340 HI=INT((SCREENMEM+40)/256)
350 LO=(SCREENMEM+40)-(HI*256)
360 POKE 87,5
370 POKE 88,LO:POKE 89,HI
390 FOR Y=9 TO 26
395 COLOR 1
400 PLOT 20,Y:DRAWTO 60,Y
410 NEXT Y
```

```
420 REM PRINT IN MODE 2
430 HI=INT((SCREENMEM+800)/256)
440 LO=(SCREENMEM+800)-(HI*256)
450 POKE 87,2
460 POKE 88,LO:POKE 89,HI
465 POSITION 0,0
470 PRINT #6;"       IN MODE 3"
500 GOTO 500
```

13.
```
10 REM CHALLENGE # 13
15 GRAPHICS 0
17 SETCOLOR 2,4,2
20 DLIST=PEEK(560)+PEEK(561)*256
30 REM 3*8 BLANK LINES
40 POKE DLIST,112
50 POKE DLIST+1,112
60 POKE DLIST+2,112
70 REM SET LMS OPTION
80 POKE DLIST+3,71
90 REM SCREEN MEMORY LOCATION
100 POKE DLIST+4,PEEK(88)
110 POKE DLIST+5,PEEK(89)
120 REM MODE 2 LINES
130 FOR X=6 TO 9
140 POKE DLIST+X,7
150 NEXT X
160 REM MODE 0 LINE
170 POKE DLIST+10,2
180 REM MODE 2 LINE
190 POKE DLIST+11,7
200 REM MODE 0 LINE
210 POKE DLIST+12,2
220 REM MODE 2 LINES
230 FOR X=13 TO 17
240 POKE DLIST+X,7
250 NEXT X
260 REM SET JUMP INSTRUCTION
270 POKE DLIST+18,65
280 REM BEGINNING OF DISPLAY LIST
290 POKE DLIST+19,PEEK(560)
300 POKE DLIST+20,PEEK(561)
305 REM WRITE TO SCREEN
310 SCREENMEM=PEEK(88)+PEEK(89)*256
```

```
320 HI=INT((SCREENMEM+140)/256)
330 LO=(SCREENMEM+140)-(HI*256)
340 POKE 88,LO:POKE 89,HI
370 POKE 87,2
380 POSITION 0,0
390 PRINT #6;"      MIXED MODES"
500 GOTO 500
```

14.

```
100 REM PLAYER 0,1,2, AND 3
200 REM SET PLAYER/MISSILE BASE ADDRESS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 1+16
300 REM CLEAR PLAYER/MISSILE RAM
310 FOR X=512 TO 1023
320 POKE PMBASE*256+X,0
330 NEXT X
400 REM PUT DATA IN PLAYERS
410 FOR PLAYER=0 TO 3
420 FOR LINES=44 TO 84
430 POKE PMBASE*256+PLAYER*128+512+LINES,255
440 NEXT LINES
450 NEXT PLAYER
500 REM SET COLORS
510 POKE 704,30
520 POKE 705,66
530 POKE 706,66
540 POKE 707,30
600 REM SET HORIZONTAL POSITION
610 POKE 53248,60
620 POKE 53249,70
630 POKE 53250,170
640 POKE 53251,180
700 REM SET VERTICAL RESOLUTION
710 POKE 559,46
800 REM ENABLE THE PLAYERS
810 POKE 54279,PMBASE
820 POKE 53277,3
850 POSITION 6,8:PRINT #6;"PLAYERS"
860 POSITION 8,11:PRINT #6;"AND"
870 POSITION 6,14:PRINT #6;"MISSILES"
900 GOTO 900
```

15.
```
100 REM PLAYER 0,1, AND 2
200 REM SET PLAYER/MISSILE BASE ADDRESS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 0
300 REM CLEAR PLAYER/MISSILE RAM
310 FOR X=512 TO 896
320 POKE PMBASE*256+X,0
330 NEXT X
400 REM PUT DATA IN PLAYERS
410 FOR PLAYER=0 TO 2
420 FOR LINES=30 TO 40
430 POKE PMBASE*256+PLAYER*128+PLAYER*20+512+LINES,255
440 NEXT LINES
450 NEXT PLAYER
500 REM SET COLORS
510 POKE 704,66
520 POKE 705,30
530 POKE 706,158
600 REM SET HORIZONTAL POSITION
610 POKE 53248,105
620 POKE 53249,105
630 POKE 53250,105
650 REM SET PLAYER SIZE
660 POKE 53256,3
670 POKE 53257,3
680 POKE 53258,3
700 REM SET VERTICAL RESOLUTION
710 POKE 559,46
800 REM ENABLE THE PLAYERS
810 POKE 54279,PMBASE
820 POKE 53277,3
850 POSITION 17,4:PRINT "RED"
860 POSITION 15,9:PRINT "YELLOW"
870 POSITION 16,14:PRINT "BLUE"
900 GOTO 900
```

16.
```
10 REM MOVE CHARACTER IN GRAPHICS 2
20 GRAPHICS 2
30 FOR P=0 TO 18
40 POSITION P+1,0:PRINT #6;CHR$(48)
50 POSITION P,0:PRINT #6;CHR$(32)
60 FOR WAIT=1 TO 100:NEXT WAIT
70 NEXT P
```

17.
```
100 REM PLAYER/MISSILE ANIMATION
200 REM BUILD PLAYER AND SET REGISTERS
210 PMBASE=PEEK(106)-4
220 POKE 106,PMBASE-1
230 GRAPHICS 3+16
240 FOR X=512 TO 768
250 POKE PMBASE*256+X,0
260 NEXT X
265 FOR PLAYER=0 TO 1
270 FOR X=20 TO 30
280 POKE PMBASE*256+PLAYER*128+PLAYER*30+512+X,255
290 NEXT X
295 NEXT PLAYER
300 POKE 704,30
305 POKE 705,66
310 POKE 559,46
320 POKE 54279,PMBASE
330 POKE 53277,3
400 REM ANIMATE PLAYER
410 J=40:K=40
415 I=1:H=2
420 POKE 53248,J
430 POKE 53249,K
440 J=J+I:K=K+H
450 IF J>208 THEN I=-I
460 IF K>208 THEN H=-H
470 IF J<40 THEN I=-I
480 IF K<40 THEN H=-H
490 GOTO 420
```

18.
```
100 REM MODIFY ATARI CHARACTER SET
200 REM RESERVE CHARACTER SET RAM
210 CHARSET=PEEK(106)-4
220 POKE 106,CHARSET-1
230 GRAPHICS 0
300 REM COPY ATARI CHARACTER SET
310 FOR X=0 TO 1023
320 POKE CHARSET*256+X,PEEK(224*256+X)
330 NEXT X
400 REM MODIFY THE LETTER A
410 FOR X=0 TO 7
420 READ N:POKE CHARSET*256+264+X,N
430 NEXT X
```

```
480 DATA 0,240,240,216,204,254,198,0
500 REM SET CHARACTER BASE POINTER
510 POKE 756,CHARSET
```

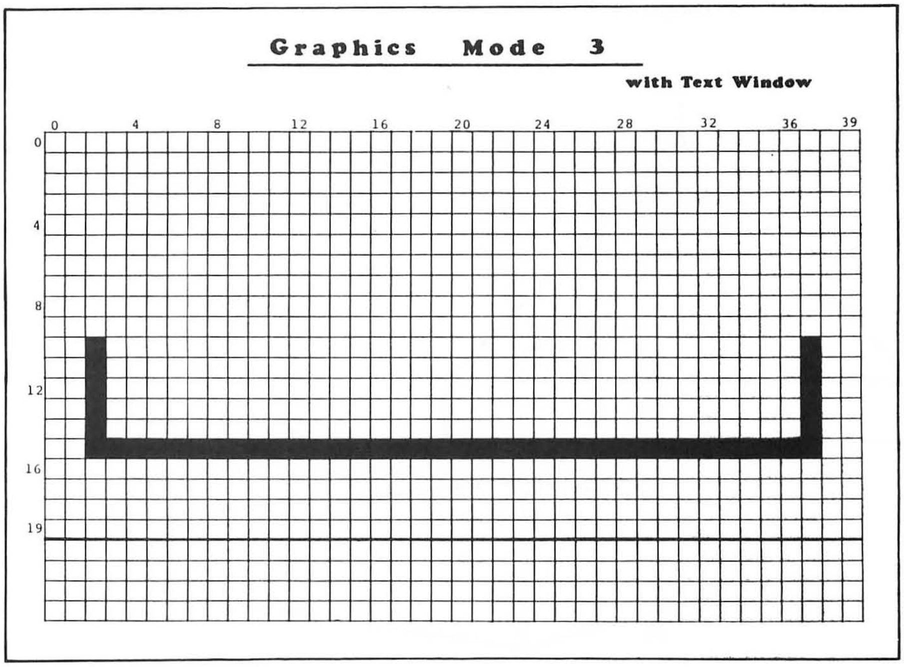
19.
```
100 REM CHARACTERS IN GRAPHICS MODE 6
200 REM SET GRAPHICS MODE
210 GRAPHICS 6+16
220 DIM CHAR$(10)
300 REM FIND SCREEN RAM POSITION
310 SCREENRAM=PEEK(88)+PEEK(89)*256
400 REM SET CHARACTER STRING
405 PX=9:PY=35
410 CHAR$="A":GOSUB 500
415 PX=7:PY=55
420 CHAR$="SQUARE":GOSUB 500
450 REM DRAW SQUARE
460 COLOR 1
470 PLOT 40,20:DRAWTO 120,20:DRAWTO 120,80:DRAWTO 40,80
:DRAWTO 40,20
490 GOTO 490
500 REM POKE CHARACTERS ON SCREEN
510 FOR U=1 TO LEN(CHAR$)
520 I=224*256+((ASC(CHAR$(U,U))-32)*8)
530 J=SCREENRAM+PY*20+PX+U-1
540 FOR Z=0 TO 7:POKE J+Z*20,PEEK(I+Z):NEXT Z
550 NEXT U
560 RETURN
```

APPENDIX C

Chapter Review Answers

Chapter 1

1. GRAPHICS
2. SETCOLOR
3. PLOT
4. COLOR
5. DRAWTO
6. POSITION
7. PRINT
8. Pink
9. Green
10. Orange
11.



Chapter 2

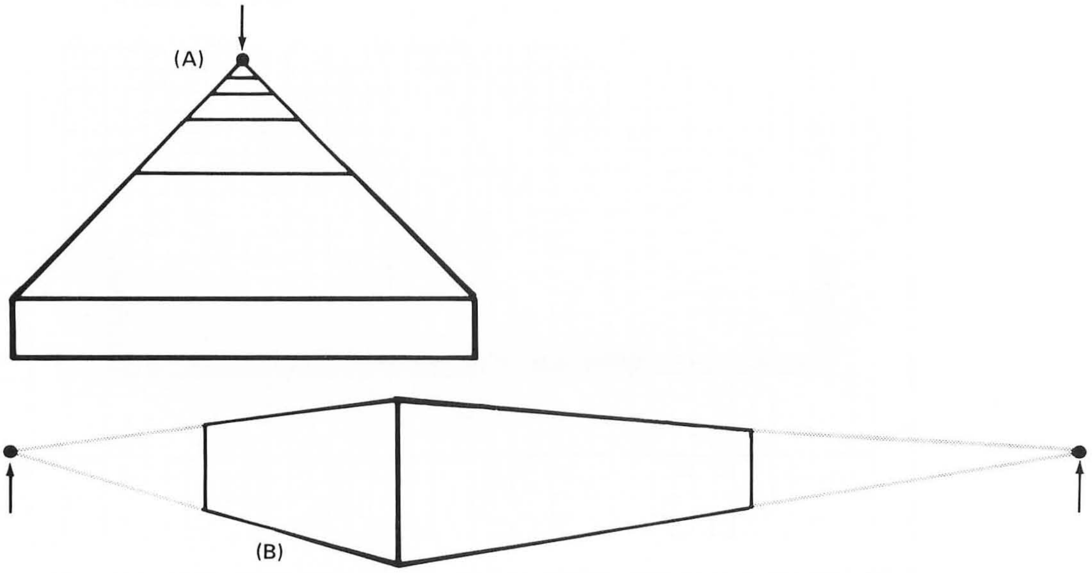
1. B
2. A
3. E
4. D
5. C
6. 4
7. 4
8. 0
9. 4
10. 2
11. 2
12. 4
13. Orange
14. Blue
15. Pink

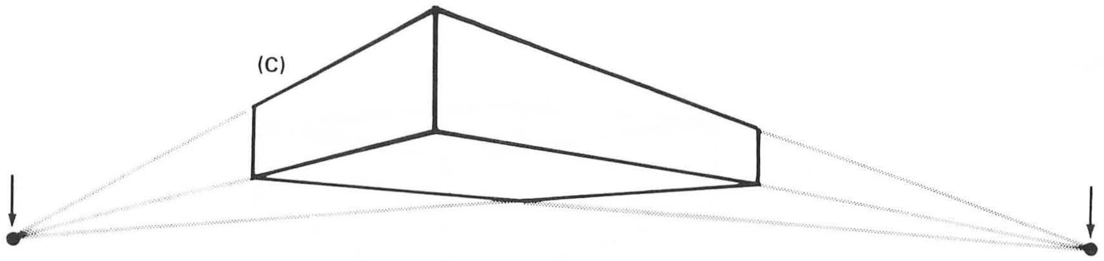
Chapter 3

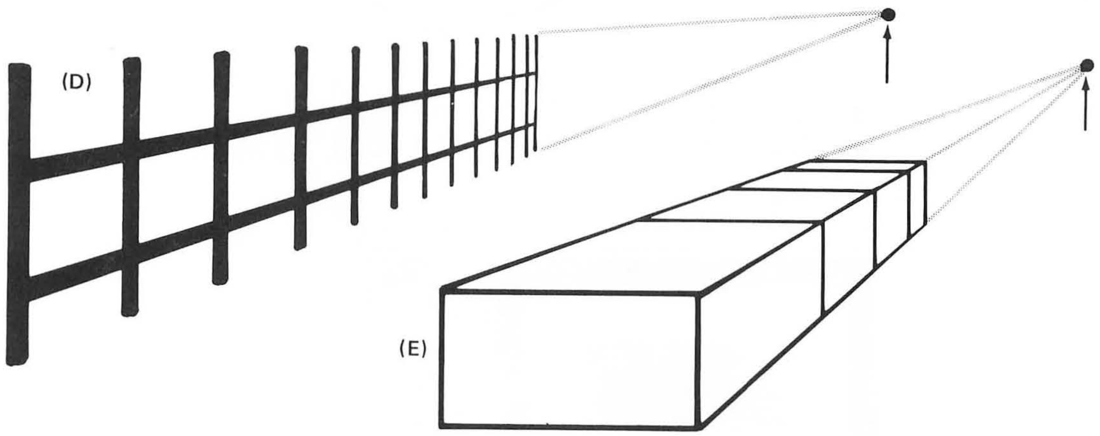
1. Red-blue
2. Yellow and orange
3. Blue
4. Warm
5. Mode 9
6. 22
7. A) Blue
   B) A vase
8. A) Warm
   B) Orange
9. Green and Blue
10. Blue and Red
11. Orange

Chapter 4

1.



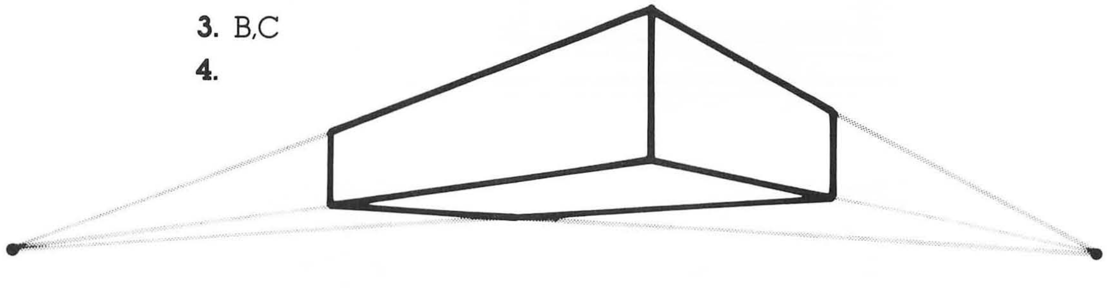




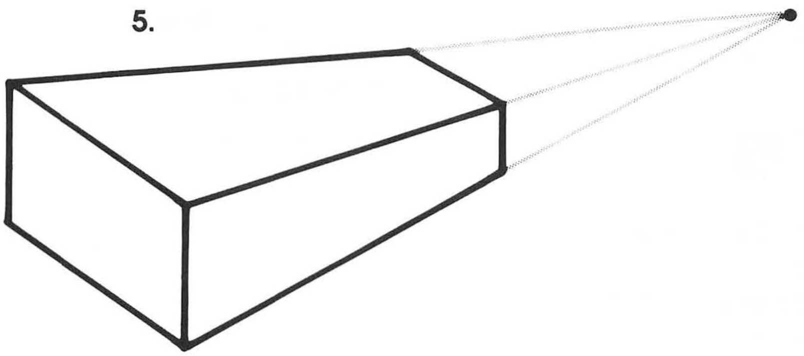
2. A,D,E

3. B,C

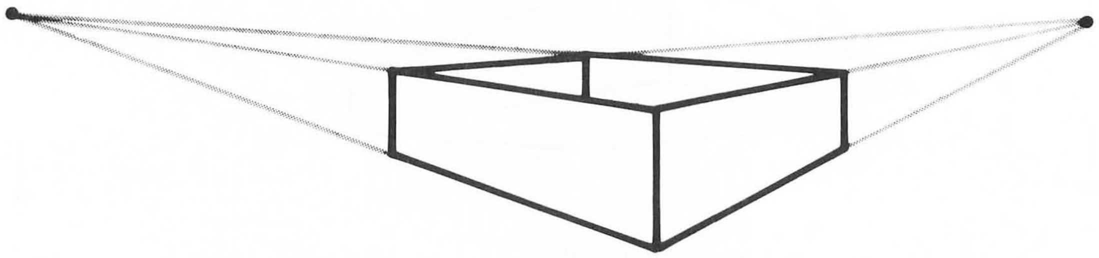
4.



5.

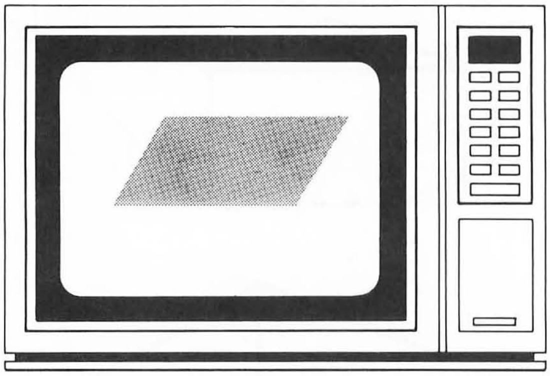


6.



7. a) 11,16 b) 19,11 c) 25,7 d) 28,5
e) 33,2 f) 32,5 g) 31,7 h) 30,11 i) 28,16

8.



Chapter 5

1. B

2. D

3. A

4. C

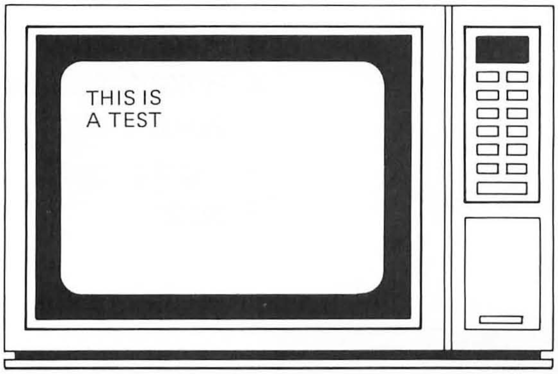
5. H
▨
F

6. a) Dark blue
b) 2
c) Light green
d) 1
e) Orange
f) 0
g) Red
h) 3
i) Light green
j) 1

7. POKE 83,22

8. POKE 752,1

9.



```
THIS IS
A TEST
```

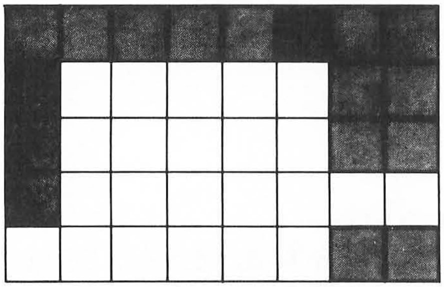
Chapter 7

1. 32 scan lines
2. 44 GRAPHICS 7 mode lines
3. Four GRAPHICS 2 mode lines, sixteen GRAPHICS 5 mode lines, thirty-two GRAPHICS 7 mode lines.
4. 87 (decimal)
5. High byte = 55 (decimal)
   Low byte = 216 (decimal)
6. 20 bytes
7. 560 holds the low byte.
   561 holds the high byte
8.
```
112                         7
112                         7
112                         7
66                          7
PEEK(88)                    65
PEEK(89)                    PEEK(560)
2                           PEEK(561)
7
7
7
7
7
7
7
```

Chapter 8

1. 120
224
224
255
7
15

2.



3. POKE 706,132

4. POKE 53250,114

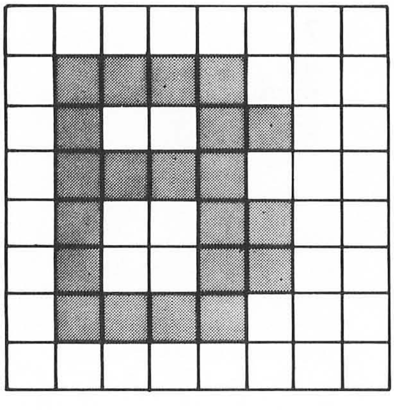
5. POKE 53258,1

6. Eight pages

7. Bit 2

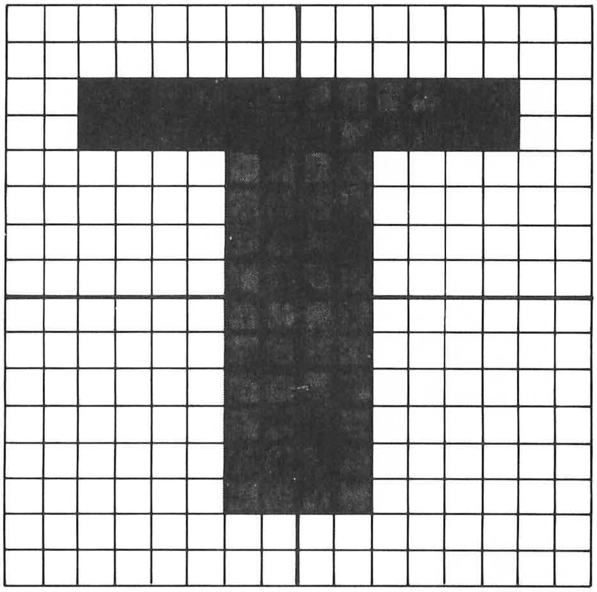
Chapter 10

1.

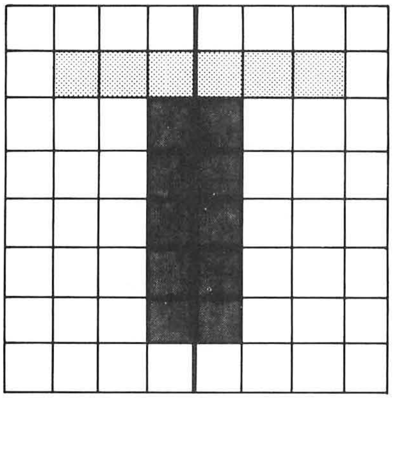


128	64	32	16	8	4	2	1	Decimal Value
0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	120
0	1	0	0	1	1	0	0	76
0	1	1	1	1	0	0	0	120
0	1	0	0	1	1	0	0	76
0	1	0	0	1	1	0	0	76
0	1	1	1	1	0	0	0	120
0	0	0	0	0	0	0	0	0

2.



3. 8 bytes
4. 1024 bytes
5. 4 — 20
   S — 51
   w — 119
6. 56864
7. SCREENRAM + 20
8.



Left Byte	Right Byte
0	0
63	252
3	192
3	192
3	192
3	192
3	192
0	0

Bibliography

Itten, Johannes. THE ELEMENTS OF COLOR. Van Nostrand Reinhold Company, 1970.

Rowley, Thomas E. ATARI BASIC—LEARNING BY USING, W. Hofacker GMBH, 1981.

ATARI BASIC REFERENCE MANUAL. Atari Inc., A Warner Communications Company, 1980.

Crawford, Winner, Cox, Chen, Dunion, Pitta, Fraser, Makreas. DE RE ATARI. Atari Program Exchange, Atari Inc., 1981.

Index

Tom Rowley

DESIGNS FROM YOUR MIND with ATARI® GRAPHICS

Whether we are aware of it or not, we are all artists and have accumulated a store of unique experiences and perceptions ready to be translated into creative expressions.

Tom Rowley has found the key to release those creative urges. *Designs From Your Mind* will allow you to realize your imaginative visions with the computer, just as the artist uses canvas and paint.

This book is a tutorial divided into two parts: Part One is an introduction to shapes, colors, and screen design, and does not require an extensive computer background. Advanced graphics are covered in Part Two. These techniques are specifically related to ATARI® graphics and the internal architecture of the ATARI® computer. Although it is a continuation of Part One, a knowledge of BASIC programming is essential and a knowledge of assembler programming is helpful.

Designs From Your Mind provides hours of fun for both the novice and experienced programmer.

The Table of Contents includes:

RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

0-8359-1276-0

Cover Illustration by: Donna Ward
Cover Design by: Joyce Thompson