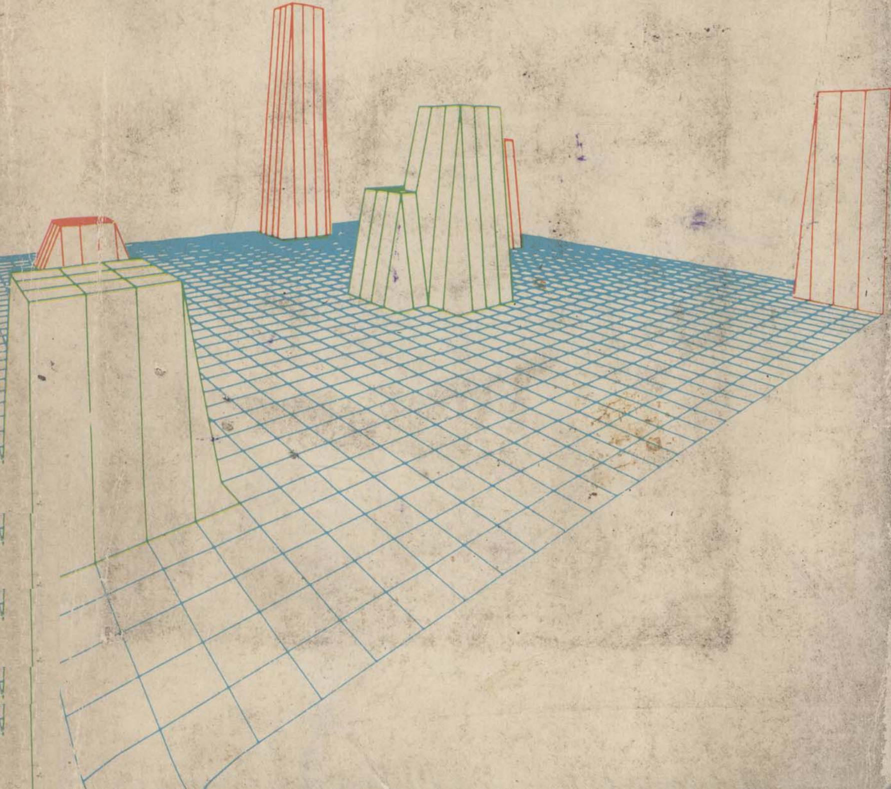
TAB 1276 $9.95

COMPUTER GRAPHICS with 29 ready-to-run programs

by David Chance

An exciting collection of entertaining and educational games and graphics . . . especially for computer buffs just getting started!



COMPUTER GRAPHICS

with 29 ready-to-run programs

Dedication:

This book is dedicated to all TRS-80™ owners.

COMPUTER GRAPHICS with 29 ready-to-run programs

by David Chance

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

1165

FIRST EDITION

THIRD PRINTING

Library of Congress Cataloging in Publication Data

Chance, David.
Computer graphics—with 29 ready-to-run programs.

Includes index.
1. Computer graphics. I. Title.
T385.C45 001.64'43 80-28652
ISBN 0-8306-9636-9
ISBN 0-8306-1276-9 (pbk.)

Cover art courtesy of Computer Graphics by Melvin L. Prueitt,
Dover Publications, Inc.

Contents

Preface

If you aren't using graphics with your TRS-80 microcomputer you are leaving out the most fascinating aspect of your computer. Whether playing, experimenting or running a program, graphics opens hidden areas of your computer.

This book contains 29 programs that are tested and ready to run. Each program uses graphics. The first chapter helps you to get started. Several different program examples are given that you can experiment with. Chapter 2 shows you how to generate graphics patterns and the random and controlled movement of objects. Program examples illustrate INKEY$, PEEK and POKE.

If you are just learning about graphics, this book is ideally suited for you. The beginning chapters are written in layman's terms to avoid confusion. If you are already using graphics on your computer you might find the sample programs quite useful and you might still learn something you didn't know.

I might stress that if you are a beginner you should read the first few chapters carefully before running the programs contained in Chapter 3. This will give you a firm understanding about what goes on with each program.

Each program in Chapter 3 has a flowchart, plus notes on the changes you might make in the programs (to experiment). The notes will step you through each program line (unless the line is self-explanatory). Each program contains REM STATEMENTS to help you understand the workings of the program. The print statements are 38 characters long to suit the book's page width. Rewrite them to increase the readability of your program.

David Chance

Chapter 1
Getting Started

Getting started is one of the easiest things to do when working with graphics. The most advantageous programming aids you can have are called Video Worksheets (catalog number 26-2105). You can buy them at any local Radio Shack store. One side of these worksheets contains all the TAB locations, PRINT@ locations and the X and Y locations of your video display. These worksheets are invaluable when you want to set up a specific display containing many different locations. On the opposite side of these worksheets you'll find numerous spaces for program lines, your variable list and comments. All program lines can be kept neatly along with your variable list in the squares provided.

After you have a general idea of the graphic display you want, chart it down on the worksheets and at the same time fill in the necessary lines and variables.

If you are new at creating graphics for the TRS-80 you will soon learn that one thing you cannot create is an ordinary circle (other than the letter O and the zero). So if you're out to create a bunch of circles on the display, you've chosen the wrong computer.

There are, however, thousands of creations that can be made with the TRS-80. If speed of execution isn't a factor, you can always rely on the SET-RESET function. This is the best route if you're just learning about graphics. The video display has 6144 graphic locations arranged in a 128 × 48 matrix, that can be Set and Reset at your command. That many locations provide you with many different creations waiting for you to draw. Just playing with the TRS-80 will open up many new frontiers.

You can use the following functions to set up the display:

1. CHR$
2. PRINT@
3. STRING$

CHR$

The following program lets you examine the graphics codes:

```
 10  CLS
 20  A=0:REM PRINT @ LOCATION
 30  FOR CODE=129 TO 191
 40  PRINT@ A,CODE;CHR$(CODE)
 50  IF A=896 THEN 70:ELSE A=A+128:REM SPACE TO
     EXAMINE CODES
 60  NEXT:GOTO 90:REM END OF RUN
 70  PRINT@ A+20, "PRESS ENTER TO CONTINUE";:
     INPUT X
 80  A=0:CLS:GOTO 60:REM RESET A - CONTINUE LOOP
     OF CHR$
 90  PRINT@ A+20,"PRESS ENTER WHEN FINISHED";
100  INPUT X
110  CLS:END
```

Line 20 starts the PRINT@ location at 0 (zero).
Line 30 FOR NEXT loop for graphic codes 129 to 191.
Line 40 prints the code and sets the graphic area.
Line 50 checks variable A before adding a space.
Line 60 the NEXT for continuing and end of program to Line 90.
Line 70 is for user to examine codes before clearing video.
Line 80 resets A to 0 (zero) and continues LOOP.
Line 90 terminates program when ENTER is pressed.

This program will let you more closely examine the graphic codes and lets you know just exactly where these blocks will be set on a line (horizontal lines 0 - 15). An example: Note that graphic code 131 produces a square block set at the top of the line, or at the top of a graphic cell (made up of 7 bits). Code 140 locates the square at the bottom of the line (or graphic cell).

What's the point? If you've created any form of graphic display using the CHR$ mode, you've probably found that to get the display you want, it can become quite tedious not knowing exactly where the graphic block will be set on a given line. By running the above program (and saving it on tape for future reference) you'll have a better knowledge of these graphic codes.

Naturally the CHR$ function is also useful for other purposes, but this is a graphics book and everything can't be taught here.

PRINT @

The PRINT @ statement is a very useful function when working with computer graphics. A total of 1024 PRINT@locations are located on the video display—a more than ample amount. By following the video-display worksheet you can place characters exactly where you want them (unless you're a pro and know all locations by heart). You have to be careful not to let the characters fall to the next line when setting up a display. For example, PRINT @ 57,"COMPUTER", would put "COMPUT"on the top line, while the R to COMPUTER would fall to the next line. If you are working on a program that contains both graphics and characters (words) be sure to add the trailing semicolon, or you'll end up with a carriage return blanking-out part of your display.

You can achieve some very pleasing displays when PRINT@ and CHR$ are combined (combined as was the previous program). When placed properly, these two functions make large letters. It's very simple! Examine the graphic codes, chart each one on a video worksheet and zap(!). You have an instant bulletin board. If you have pre-schoolers, these large letters could be their road to learning the alphabet. The bulletin-board program could be constructed to print out sentences using the large letters. A FOR-NEXT loop would create a delay between words.

Your imagination is the only limit to the things you can do with a computer (any computer). If you don't know everything about your computer, playing with it serves as an excellent teacher.

STRING $

The STRING$ statement (or function) is something else that is very useful when creating graphics. The only drawback to using this statement is that unless you clear enough memory before you use it you'll end up with an OS ERROR (out of string space).
An example using the STRING$ statement:

```
10  CLEAR 64:CLS
20  A$=STRING$(64,43)
30  PRINT A$
40  GOTO 40
```

Line 10 clears just enough memory (string space) for the STRING$ statement in line 20. Of course if you were to see a number larger than 64, you would have to clear more memory.

Line 20 contains the actual STRING$ statement—STRING$(64,43). 43 is the character code representing the plus sign and 64 is the number of plus signs that will be printed by line 30.

If you were to change the character code from 43 to the graphic code 191 (for all bits on), the computer would print a solid bar across the top of the video. Using the short program above and graphic codes 129 through 191, experiment with the program.

CHR$, PRINT AND STRING$ (COMBINED)

Combining these three statements will give you an array of all the possible patterns on your display. For example:

```
10   CLEAR 1000:CLS
20   A$=STRING$(63,191)
30   B$=CHR$(149)
40   PRINT @ Ø, A$
50   PRINT @ 96Ø, A$;
60   FOR L=64 TO 896 STEP 64:PRINT @ L,B$;:NEXT
70   FOR R=126 TO 958 STEP 64:PRINT @ R,B$;:NEXT
80   GOTO 80
```

The above program would print a neat border around the outer edges of your video display. Code 149, (line 30) just turns on three bits of a line. Step 64 in lines 60 and 70 keeps the left and right lines vertical. Experiment with the program by inserting different values for A$ and B$, and you'll come up with some rather unusual displays. Be sure to add the trailing semicolon to keep from getting a carriage return.

The program VIEWER contained in this book will further your knowledge of the differences between, SET-RESET, CHR$, PRINT @, STRING$ and POKE. The program merely sets a display and shows the different operating speeds of each function. You'd be surprised at how slow POKE is compared to using CHR$, PRINT @ and STRING$ for the same program.

Chapter 2
Generating Game Patterns

You can generate game patterns in many different ways. The choice is entirely up to you. If you're concerned about the program's execution speed you should probably POKE your graphics; but, as was said in the last chapter, execution speed can be obtained from other functions. Besides, the old adage "if you don't know where you are POKEing *don't,*" is very true.

Your best bet would be to start with some other function until you become more familiar with the hazards of POKE (explained later in this chapter). We use SET and RESET extensively throughout the programs in this book. Why? So the reader can become more acquainted with its function. Thousands of games can be generated using the SET-RESET function. They won't be the fastest, but they'll be easier to write and handle. As stated earlier, you can also generate game patterns using the CHR$, PRINT @ and STRING$ function. In my opinion (and only mine) top execution speed comes from using the above functions. If you still are hesitant to believe me, and haven't done it yet, key in and run the program VIEWER, then see what you think.

Generating game patterns can also be done randomly. This is the trick to the program MOUNTAIN. Arguments placed within the program keep the RANDOM generation within certain limits. Always remember to place your arguments *before* the RANDOM statement, placing them afterwards will only lead you to a FC ERROR (Illegal function call). This means simply that the statements have gone beyond the set parameters that the computer and video display are capable of handling.

RANDOM MOTION

You might want to move an object randomly. For example, a program might contain some spacecraft that you want moved about the video screen randomly.) An example might be:

```
 10  CLS
 20  RANDOM
 30  C$="<:<0>:>":N$="           "
 40  A=410
 50  PRINT@ A, C$;
 60  FORI=1 TO 50:NEXT
 70  PRINT@ A, N$;
 80  M=RND(2):ON M GOTO 90,100
 90  IF A > =440 THEN 40:ELSE A=A+RND(4):GOTO 50
100  IF A< =380 THEN 40:ELSE A=A–RND(4):GOTO 50
```

Line 10 clears the video screen.

Line 20 seeds the RANDOM generator.

Line 30 of course is the spacecraft (C$). N$ must contain the same amount of spaces that C$ has or parts of the spacecraft will be left at different locations on the video.

Line 40 variable A is the starting PRINT @ area.

Line 50 prints the spacecraft at A.

Line 60 is a time loop before the spacecraft is blanked-out.

Line 70 prints N$ at A to blank-out spacecraft before variable A is changed.

Line 80 a random function either selects left or right movement of the spacecraft.

Line 90 has the argument placed *before* A is increased so the craft won't fall to the next line. If variable A is greater than or equal to 440, the craft will return to the beginning position and A is reset to 410.

Line 100 is the exact opposite of line 90, meaning the craft will be moved to the left.

Both lines 90 and 100 return (GOTO) line 50 to keep the craft moving.

If you've entered and run the preceeding program you should have noted that experimenting with the time loop, (line 60) lets you move the spacecraft at a slower or faster rate. You can also cause the craft to jump more or less by changing the RND function in lines 90 and 100. By adding several more lines and arguments, you can actually move the craft up, down left or right.

You are probably wondering now, what is the point of all these games? The point is that you can learn the many different ways in which your computer is capable of operating by just running several different games. Some games contain most of the functions that a computer has. And what better way to relax after a hard day than powering up a computer and running a game? Sitting in front of the T.V. watching reruns made two decades ago?

Back on the track . . . . . You can apply the same random motion with the SET-RESET function. Enter the following short program and run it:

```
100  CLS
110  RANDOM
120  X=64:Y=47
130  IF X< =0 OR X> +127 THEN 120:ELSE SET(X,Y)
135  FOR I=1 TO 10:NEXT
140  IF Y< =0 THEN RESET(X,Y):GOTO 120
150  RESET(X,Y):Y=Y-1
160  M=RND(2):ON M GOTO 170,180
170  X=X+RND(5):GOTO 130
180  X=X-RND(5):GOTO 130
```

Line 120 sets the area for X and Y to begin.

Line 130 tests X before setting X and Y (if argument is true X and Y will be reset to the beginning position).

Line 135 is a short loop to slow down the SET & RESET of the light block area.

Line 140 tests Y, if true, RESETS X & Y and goes back to the starting position (line 120). If RESET were left out of this line the block areas would remain light at different points, top of video.

Line 150 RESETS X & Y also desends Y, so block will move to top of video.

Line 160 almost exactly as line 80 in the last program, but selects (randomly) either left or right movement of the block.

Line 170 & 180 are for right or left movement of the block respectively and returns back to line 130, to keep the block moving.

With minor program modifications the light block could be moved up, down, along with the right and left movement. With the addition of more variables you could have more than one block moving around the video.

CONTROLLED MOTION

Once you have an object moving randomly on the screen, you take the program with the spacecraft and add another dimension to it, such as a guided missile.

```
 10  CLS
 20  RANDOM
 30  C$="<:<0>:>":N$="          ":S=989:M$=CHR$(143)
 35  PRINT@ S, M$;:FOR X=0 TO 127:SET(X,47):NEXT
 40  A=410
 50  PRINT@ A,C$;
 60  FOR I=1 TO 50:NEXT
 70  X$=INKEY$:IF X$=" " THEN 140
 90  PRINT@ A,N$;:M=RND(2):ON M GOTO 100, 110
100  IF A > =440 THEN 40:ELSE A=A+RND(2):GOTO 120
110  IF A < =380 THEN 40:ELSE A=A-RND(2)
120  PRINT@ A,C$;
130  IF X$< > "   " THEN 50
140  PRINT@ S, "   ";:S=S-64:PRINT@ S,M$;
150  IF S=(A+3) THEN PRINT @ 25, "** A HIT **":FOR I=1
     TO 500:NEXT:GOTO 170:ELSE IF< > 349 THEN 90
170  PRINT@ 25,"        "
180  PRINT@ S,"   ";:S=989:PRINT@ S,M$;:GOTO 50
```

Note that you've made only a few minor changes. At line 30 M$ is for the rocket and variable S is the rocket's area.

Line 35 prints the rocket at S.

Line 70 is the INKEY$ function, " " meaning, press the space bar to fire.

Line 130 checks X$ to see if rocket has been fired.

Line 140 fires the rocket (that is, if the space bar was pressed). Prints a blank, descends S by 64, then prints the square? rocket again. And, while the rocket is moving in an upward motion:

Line 150 tests S, to see if at a certain point S will equal A+3 (A+3 is the center of the spacecraft, where the 0 is located). If it does, line 150 will prints "** A HIT **" @ 25, goes into a time loop, (so message can be seen) drops down to line 170 and recycles. If you miss the spacecraft, or S <> 349, line 150 returns to line 90. This causes the 'flutter' of spacecraft. If S does equal 349, S is reset and returns back to its starting position (989).

With several other modifications to the above program could provide more than one spacecraft, allow the rockets to fire in different directions and, of course, add scoring.

You can also use motion control to manipulate things other than rockets. An example might be moving words (horizontally) across the video screen.

Enter and run the following example:

```
 10  CLS:PRINT CHR$(23)
 20  A$="THEY"
 30  B$="WENT"
 40  C$=LEFT$(A$,2)+"A"+RIGHT$(B$,1)
 50  D$=LEFT$(B$,1)+"A"+RIGHT$(A$,1)CHR$(94)
 60  FOR W=64 TO 120:PRINT @ W,A$;:PRINT @W-1,
     "  ";:NEXT
 70  FOR W=128 TO 184:PRINT @ W,B$;:PRINT @
     W-1,"  ";:NEXT
 80  FOR W=192 TO 248:PRINT @ W,C$;:PRINT @
     W-1,"  ";:NEXT
 90  FOR W=256 TO 312:PRINT @ W,D$;:PRINT @
     W-1,"  ";:NEXT
100  GOTO 100
```

Line 10 clears the video screen and changes to the 32 character-per-line format.

Lines 20 through 50 contain the four words that are used.

Lines 60 through 90 sweep each word across the video screen from left to right. The statement PRINT @ W-1, " "; is used to blank-out letters to the left of the words. If that statement were to be left out, it would defeat the purpose of the program.

By adding a few more lines, you could have the words bounce off the right side of the video screen and return to the left side (inserting STEP-1 statements and change numerical contents of the FOR NEXT loops). You would also have to add a PRINT @ statement, W+1," "; to blank-out letters on the right side of the words.

INKEY$

Just about the most common way to control objects on your video screen is to use the INKEY$ function. The INKEY$ function allows you to use just about any key desired to control an object. You must be sure to insert the INKEY$ statement within a specified line, however. For example:

```
10  X$=INKEY$:REM SETTING X$ TO THE INKEY$
    FUNCTION
20  PRINT @ 64,X$
30  GOTO 20
```

Obviously inserting the above lines in a program won't work! You could punch keys until your fingers fell off, and nothing would be printed at 64. The program runs so fast that all it would be doing is going from 20 to 30 and back again. What line 30 should have read was:

30 GOTO 10

This way you could press all the keys you wanted and each key pressed would be printed at 64. Anytime you are using the INKEY$ function you must have, X$=INKEY$ (X$ is only an example) located where the computer will recognize what X$ is for. Otherwise, you'll end up having problems with your program.

If you plan on making a program that will use the INKEY$ function and would print the letters at a specified area in order to create a word, you might try the following example:

```
 10  CLEAR 500: CLS
 20  A=64: WW=0
 30  X$=INKEY$: IF X$="" THEN 30
 35  IF X$="  " THEN 80: ELSE IF X$="0" THEN 120
 40  W$=W$+X$
 50  PRINT A,X$;
 60  A=A+1: WW=WW+1
 70  GOTO 30
 80  A=A-1: WW=WW-1
 90  W$=MID$(W$,1,WW)
100  PRINT @ A,"";
110  GOTO 30
120  CLS:PRINT @ 128, W$
130  END
```

This program example will receive whatever characters you press on the keyboard (excluding Ø, which ends the program). The up arrow backspaces a character to let you erase and change it. You can also insert spaces between words by pressing the space bar.

Line 10 clears 500 bits of memory for string space.

Line 20 variable A is for character placement, variable WW is used for backspacing only.

Line 30 starts the INKEY$ function.

Line 35 contains two arguments. The first one (up arrow) is for backspacing. When the up arrow is pressed control falls to line 80 where variable A is decreased by one (one for each time the up arrow is pressed) to blank-out the last character. Variable WW is decreased by one so that the length of W$ is decreased by one, this