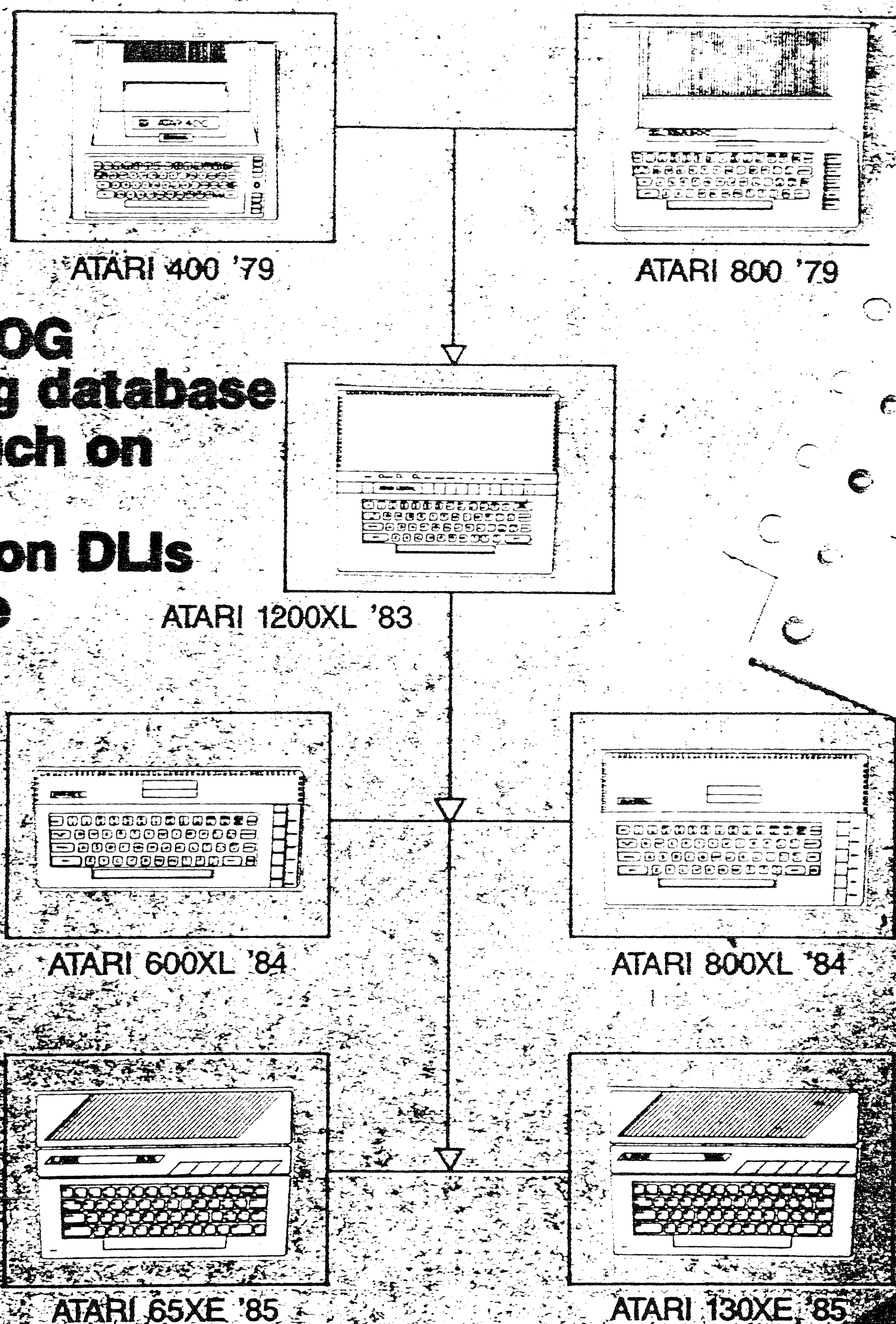
NO. 47
OCTOBER 1986

CANADA $4.75

THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

A.N.A.L.O.G. COMPUTING



The ANALOG Computing database
John Skruch on the 8-bits
A tutorial on DLIs
Deathzone

PLUS

NO. 47 OCTOBER 1986

THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

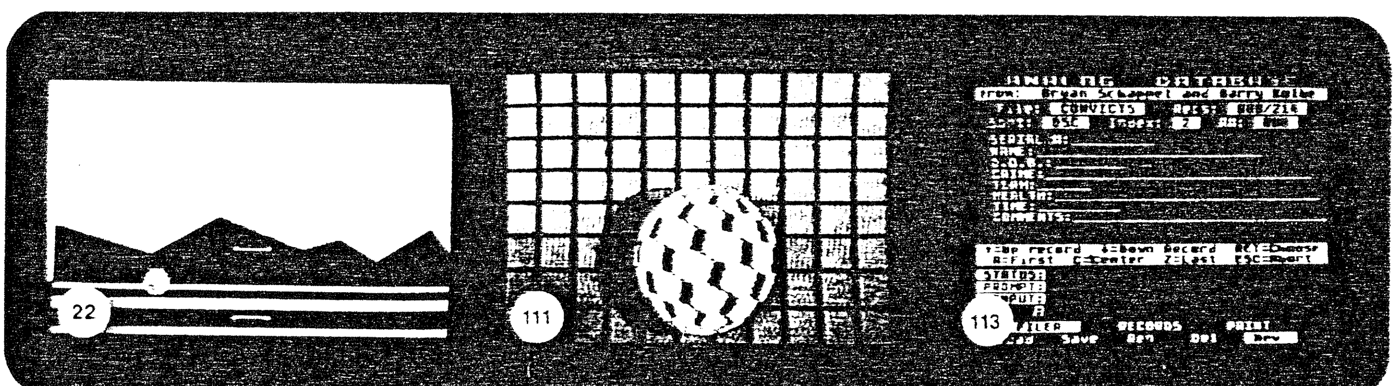
ANALOG COMPUTING

FEATURES

FEATURES *continued*

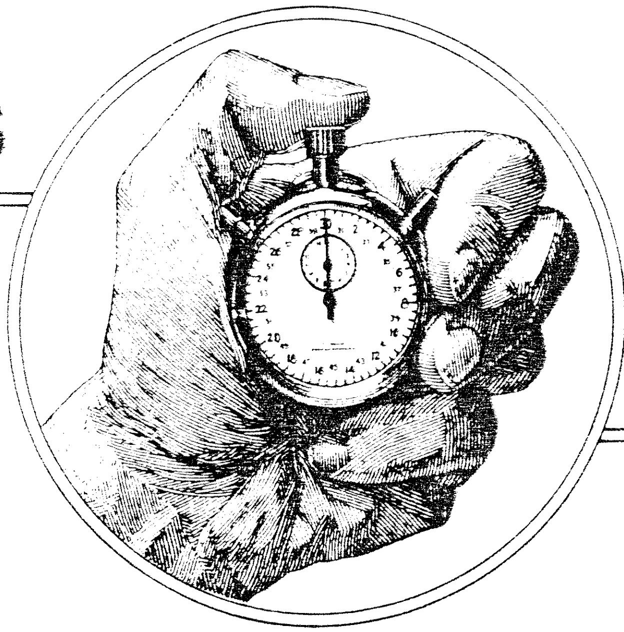
REVIEWS

COLUMNS



ANALOG Computing (ISSN 0744-9917) is published monthly for $28 ($36 in Canada, $39 foreign) per year by ANALOG 400/800 Corp., 565 Main St., Cherry Valley, MA 01611. Second class postage paid at Worcester, MA and additional mailing offices. POSTMASTER: Send address changes to **ANALOG Computing**, P.O. Box 625, Holmes, PA 19043.

DLIs



A minute to learn

by Jonathan David Farley

A wise author of a technical article (mainly, myself) once cautioned, "Highly technical jargon will always drive even your boldest readers straight up the trunks of any nearby trees." *Id est*, novices run from big words in books and magazines faster than disks from lodestones. Display list interrupts, or DLIs, however, shouldn't frighten an experienced programmer like yourself away from this month's **ANALOG Computing**.

Yes, I admit that whoever coined the term *display list interrupts* just might've made it a little more self-explanatory (or perhaps a tad shorter), so as to eliminate the need for articles such as this, but, alas, the party involved did not. That's what I'm here for, anyway—to show, tell and explain to you all there is (that I know) about these elusive, conducive, cybernetic creatures controlled via computer (and keyboard), using little or no alliteration.

Now that the introduction to this piece is done, you may ask with impunity, "What's a DLI?" I'm glad you did; a DLI is an interruption in the display list. You may ask, "What's a display list?" (Go on, ask.) Luckily for you, I have the answer; unfortunately, it's rather involved.

Watching TV.

The display list has, of course, something to do with the display produced by your TV set or monitor. Hence, to understand DLIs, you must understand your TV (or monitor). If you've ever examined the screen in a wild paroxysm of curiosity, you may have noticed it's composed of a myriad of diminutive dots. (If you've never, ever scrutinized the display, I'll wait while you do.) Each of these screen specks emits a specific color, and, from a distance, all of them form the image you see. (This operates along the same lines as the art form pointillism: a few paces from a painting, the human eye cannot differentiate two contiguous spots, so the whole picture is mentally interpreted as one image, not as multichromatic speckles.)

A special beam produced by your set gives the motes of the display their colors, sweeping over every one of them sixty times in a single second, starting from the top left corner. The beam's intensity can be controlled, to create a bright or dim glow. From the starting position, it brushes over to the right, then turns itself off, so it doesn't fill in anything on its return to the left. The special ray automatically lowers itself slightly, and goes back to the left side. Here, it restarts to color yet another line of dots, turns off on the right, drops and rushes to the left again. One horizontal line of filled specks on the screen is termed a "horizontal scan line." When the TV completes 192 of these (or thereabouts), the display is finished, and the beam resets itself to the top left of the glass.

The times when the ray is off have special names. When it's on the right and going back to the left to turn back on, it's the "horizontal blank." Since screen dots are the elements which collectively form a picture, they're called "picture elements," or "pixels."

I hope it seems obvious that the Atari computer needs some way to tell the TV what to show on the screen. Here's how it does so.

In BASIC, the screen can only have one graphics mode at a time. The screen's either graphics 2 or graphics 6 (or any of the other BASIC graphics modes), but never more than one mode at a time (most of the time). By manipulation of the display list, the display can be up to fourteen different modes on one screen!

Perhaps you think of BASIC graphics mode 0 as just that—BASIC graphics mode 0. Your Atari thinks of it as ANTIC mode 2.

ANTIC is a video microprocessor; it controls the TV display, including the display list you've heard so much about. To understand the display list, think of BASIC graphics modes not as screens (as in mode 0) of text, but as piles of lines, or "mode lines" (i.e., mode 0 is really a pile of twenty-four mode lines on top of each other). If you think of it that way, you can see how one mode line could be replaced by a different mode line. That's how the computer gets so many graphics modes on one screen.

Inseparable screen?

The display list is a series of bytes located in your computer's RAM. Its exact address can be found by multiplying the contents of location 561 (by PEEKing into it) by 256 and adding the product to the contents of 560. When you switch graphics modes, your computer creates a new display list. Type in this program:

```
10 GRAPHICS 0
20 DL=PEEK(561)*256+PEEK(560)
30 FOR A=0 TO 31
40 PRINT PEEK(DL+A);" ";
50 NEXT A
60 GOTO 60
70 REM PRESS break TO STOP ME!
80 END
```

This program PRINTs the 32 bytes of the BASIC graphics mode 0 display list onto the screen. It should start out with three 112s, a 66, followed by two other numbers, several 2s, and a 65, followed by two other numbers. Just about every number correlates to one line on your screen. For instance, I told you that BASIC mode 0 is ANTIC mode 2. Those 2s in the display list indicate mode 0 lines.

DL shift.

Since in BASIC mode 0 there are forty characters per line, when you type the graphics 0 command, the computer takes 40 bytes and puts them on one line. However, if you switch mode lines, it still thinks every mode line is ANTIC 2. The result: wait, I'll show you!

Take a BASIC mode 3 line interposed right in the middle of a graphics 0 display. Well, in a graphics 3 display, you know that you can plot up to 40 pixels across the screen. Since it only takes 1 byte to produce 4 pixels, then every line in BASIC mode 3 must take 10 bytes, as opposed to the 40 in a mode 0 display. When the computer still thinks it's in mode 0 for every line, it will take an extra 30 bytes' worth of characters after the mode 3 line of 10 bytes. It believes the line is a normal 40 bytes.

Since the 10 bytes of the mode 3 line take up one physical line (see the Atari BASIC Reference Manual for the definition), the next 30 bytes must be displayed on the next physical line. That means everything will be shifted over 30 bytes, or pixel positions. You can try it and see for yourself; just POKE the ANTIC mode value equivalent for BASIC mode 3 (8) in DL+16.

Table 1 tells you the number of pixels that can be plotted per mode line, as well as the number of bytes per line.

must use different byte groupings.

More codes of modes.

If you substitute, say, 10 for the POKE values in Lines 40 and 50, you'll see plotted pixels in the band, instead of characters. ANTIC 10, or BASIC graphics mode 5, is not a text mode.

The same byte means different things in different modes. It's just translated another way, much like a byte in BASIC 0 could be a character, but is four pixels in BASIC 3. The codes for the characters and symbols in ANTIC are given in the Atari BASIC Reference Manual, and the pixels' colors for nontext modes can be determined by using the base 10 value you find and converting it to base 2. Every pair of bytes is the color for a pixel.

Going off on a tangent once more, enter the following BASIC program:

```
10 GRAPHICS 8
20 REM BASIC GRAPHICS MODE 8=ANTIC 15
30 COLOR 1
40 PLOT 0,80
50 DRAWTO 319,80
60 END
```

The line that results from running this program is exactly one horizontal scan line. Now that you know what a horizontal scan line looks like, count how many of these there are in one ANTIC mode 2 character. If you counted eight, you're correct: every ANTIC 2 (or normal text) mode line is composed of eight horizontal scan lines. If there are twenty-four mode 2 lines in a regular graphics 0 display, the screen then has 192 scan lines. As I said before, there are only 192 effective scan lines in your screen: a graphics 0 display uses the whole screen.

Now, other modes' pixels use varying numbers of horizontal scan lines. For instance, ANTIC mode 15 pixels, as you've seen, only take up a single scan line in vertical distance, while mode 7 lines take up sixteen scan lines. Let me show you why the number of scan lines in a mode line is important—type in the previous program again.

Out of space.

In a graphics 0 display, there are twenty-four lines per screen of 40 bytes per line, or 960 per screen. After you run the program, you see twenty-two mode lines of 40 bytes each and two mode 6 lines of 20 each, for a total of 920 bytes. Where did the missing bytes go?

Actually, they didn't go anywhere. Just as the computer expects 40 bytes for every mode line, no matter what, in a graphics 0 display, it expects 960 for the whole screen. The two mode 6 lines, while taking up as many bytes as a single mode 2 line, take up twice the vertical space in horizontal scan lines.

Thus, all 192 scan lines are used up, but there's still one more line of characters left, the missing 40 bytes. Those 40 bytes are not displayed, because there's not enough space, but the computer treats them as if they were still on the display. In fact, when you move your cursor off the screen, you can't see it; it's in the "missing" line.

Empty space.

We've looked at the "mode codes" of the display list. Now let's see what the remaining bytes in it are. Starting at the top of your screen is the border.

Take a look at it and estimate its size. Doesn't it look like three (completely blank) mode 2 lines stacked on each other? That's because both mode 2 lines and these lines have eight horizontal scan lines in them. You can create lines anywhere from one blank scan line to eight.

Let's say that N is the number of blank scan lines you want. Just POKE the value 16*N-16 into the appropriate byte of the display list. The computer will go about its business as if that line (or those lines) weren't even there.

Some TV sets have "overscan"—parts of the display are lost behind the case of your TV. In a computer, no part of the screen which might hold any vital information should be off the visible portion of the screen, for unseen information is useless. The three "blank 8" lines, as they're called, push the display down enough to ensure that overscan doesn't occur.

RAM space and inanimate communication.

The next 3 bytes of the display list go together, the *66* and the two following numbers. The first byte is an operator and the next two, operands; they could be almost any values, really. Remember how locations 560 and 561 manipulated in a special way revealed the first byte of the display list? Well, the fourth and fifth bytes of an unchanged display list for graphics 0 tell us the location of the first byte in the screen RAM. In case you forgot, in the screen RAM each byte (for mode 2 lines) is a character.

In a text mode, the character symbol displayed depends on the particular byte being used. There are 960 bytes in the screen RAM for graphics 0, to correspond to the same number of characters which may be put on the display in that full-screen mode.

To simplify, let's turn the inner workings of your Atari into some dialogue. For instance, the *66* in the display list spoken of earlier essentially says to your computer's processing unit, "Hey, you, start putting those characters' bytes you got over there in the screen RAM—hmm, it's at . . . forty-thousand on the dot, that's where the screen RAM memory starts—translate 'em to characters and put those on me, starting on this line. Oh, yeah, I almost forgot, I'm an ANTIC mode 2 line."

The *66* is actually a composite of two commands of the Atari. It's broken up into a *2* and a *64* by the computer. The *2* means, "I'm a mode 2 line," and the *64* says, "Into this line and others below me (unless otherwise specified), put the data in memory locations starting with the following." The following memory location address, (the two numbers following the *66*) is the beginning of the screen RAM. The decimal address is calculated by multiplying the second number by 256, then adding the first.

The *64+2=66* command may be placed more than once in the display list. You could have one for each mode line, for instance. In that case, the characters would come from different portions in memory for every mode line. You might even combine the *64* command with a mode 8 line, for a *64+8=72* command line, or use it with almost any other mode. The screen of your Atari can be made as diverse and complex, or as simple and basic as you desire—if you know how to do it.

The operands of the commands may also be changed. Hence, your screen RAM for a mode line may be anywhere in the computer's available memory.

Rerun.

As you may have surmised by now, the 65 toward the end of the display list is an operator, or command that operates on the next two values following it, which are the location of the first byte in the display list. The display list is really a program for the ANTIC microprocessor; it needs some way to constantly run the list. The computer examines the list like a BASIC program that's run; the 65 is like a GOTO statement, and the next two numbers are the address where it must go to start the program over again.

Unit test.

I think I've given you enough information to understand the basics of display lists. You now have the power to warp the display list any way you choose, but experimentation is the only way to measure the full capability of your Atari. So go ahead, you can do it! Next month we'll tackle the display list interrupts themselves.

Jonathan David Farley has been working with computers since taking a college computer course at age nine, and is now interested in reviewing and writing software.