Behavior Research Methods, Instruments, & Computers
1984, 16 (4), 388-394

COMPUTER TECHNOLOGY

The TMS 9918A VDP: A new device for generating moving displays on a microcomputer

BENNETT I. BERTENTHAL and STEVEN J. KRAMER
University of Virginia, Charlottesville, Virginia

A procedure for creating smooth and continuous moving displays on a microcomputer using the TMS 9918A video display processor (VDP) is discussed. This processor shares all of the features of other VDPs and, in addition, enables the user to directly program smooth motion of specified graphics shapes (sprites) with multilevel pattern overlaying. The general principles for programming this video chip are described, and a machine language driver is presented. Three illustrative applications from the event perception literature—wheel-generated motion, kinetic disruption of optical texture, and biomechanical motions—are presented as demonstrations of how the unique features of this video chip can be implemented in the creation of dynamic displays.

The microcomputer is fast becoming a major tool of the visual perception laboratory (Cavanagh & Anstis, 1980). Included among its functions are the creation and rapid presentation of well-defined shapes, texture patterns, and scenes, the implementation of interactive displays, and the simple presentation of apparent-motion phenomena. One remaining function that has proved more elusive to implement involves smooth and continuous animation, especially when surfaces occlude and disocclude one another as they move. Yet, the availability of this last function is no less important than the others, since it is essential to the study of the kinetic specification of objects and their spatial layout, a topic beginning to receive increasing interest in the visual perception literature (Johansson, von Hofsten, & Jansson, 1980).

Machine language programs represent one solution to the animation problem (e.g., Jochumson, 1983), but the availability of this programming skill is fairly restricted and programming in machine language is certainly much more laborious than programming in higher level languages. Furthermore, the frequent need for hidden line and surface algorithms adds substantially to the complexity of the programming task and, more importantly, places additional demands on the relatively slow-operating microprocessor. The net result is that, even with a machine language program, the rate of movement is often slowed to a level that compromises the perception of smooth and continuous motion, especially when large shapes are involved.

In this paper, we introduce an alternative solution for producing animated displays that is more powerful than the standard machine language approach, and yet easier to implement. The major component of this solution is an integrated circuit by Texas Instruments, the TMS 9918A video display processor (VDP).[1] This chip includes certain features that make it ideally suited for creating animation. In particular, it provides the opportunity for displaying and moving sprites.

Preparation of this paper was supported in part by NICHD Grant HD-16195. We wish to thank Dennis Proffitt for his assistance in the creation of some of the figures and Marshall M. Haith for his comments on a previous version of this manuscript. Reprint requests should be addressed to: Bennett I. Bertenthal, Department of Psychology, Gilmer Hall, University of Virginia, Charlottesville, VA 22901.

SPRITES AND OTHER FEATURES OF THE TMS 9918A VDP

A sprite is a colored block shape that can be drawn on a video plane in a position given by a single pair of coordinates specifying a particular vertical and horizontal location on the screen. These graphic objects can be programmed in two sizes: 8 x 8 and 16 x 16 pixel (picture element) arrays. The size of the individual elements can be increased from 1 pixel to a 2 x 2 array of pixels through a magnification factor. User-defined shapes are created by programming a subset of the pixels in the array to appear as a particular color (15 possible colors) and setting the remaining pixels to transparent.

Animation of each shape is accomplished by simply changing the values of the x,y coordinates corresponding to the upper left-hand corner of the sprite. This procedure provides a method for quickly and smoothly moving patterns at a resolution of 1 pixel: The full screen resolution is 256 pixels horizontally and 196 vertically. In contrast, programming animation with other VDPs necessitates that a shape be drawn at one location, erased, and then redrawn at a new location. Even for those readers unfamiliar with the programming demands of the latter procedure, it should be apparent that the use of sprite graphics is the simpler of the two alternatives.

Programming with sprites also eliminates many of the difficulties encountered when overlaying a moving object on top of another surface. Each sprite occupies 1 of 32 separate video planes organized together as a set of vertically stacked display planes, not unlike 32 panes of glass sandwiched together (see Figure 1). Sprites are assigned priorities of visibility as a function of their position in depth relative to the observer. The closest plane is assigned a priority of 0, and the furthest plane is assigned a value of 31. Any time portions of two or more sprites occupy the same location on the screen, that portion of the sprite with the lower priority (i.e., higher number) will be occluded by the sprite with the higher priority. For example, notice how the front of the car is occluded by the tree in Figure 1. Also, as the car continues to move behind the tree, there will appear a progressive deletion and accretion of texture corresponding to the car. The point to be emphasized is that neither of these effects needs to be programmed; occlusion of the car occurs simply because it is composed of sprites with lower priorities than the sprites composing the tree. Thus, the unique graphics architecture of the TMS 9918A VDP enables the programmer to simulate both static and kinetic occlusion without any need for complex hidden-surface algorithms. This feature is especially useful for simulating three-dimensional displays, since it is well known from the visual perception literature that occlusion information is perceived as specifying the depth-order relations among different surfaces, especially when one surface is moving relative to another (Gibson, Kaplan, Reynolds, & Wheeler, 1968).

In addition to the first 32 sprite planes, there exist 3 additional display planes (listed in their order relative to the observer)–pattern, backdrop, and external VDP input (see Figure 1). The pattern plane operates in any of three graphics modes and one text mode. In each of the graphics modes, the screen is divided into different-sized pattern blocks ranging from the equivalent of low- to high-resolution graphics. The backdrop plane is programmed to appear in 1 of 15 colors and provides a border around the sprites and pattern plane. Finally, it is possible to input an external video signal that will appear behind the other planes. (Note–some versions of this chip allow the external signal to emanate from the host microcomputer's VDP.) In all cases, the visibility of a particular location on any one display plane assumes that all the preceding planes are transparent in that location.

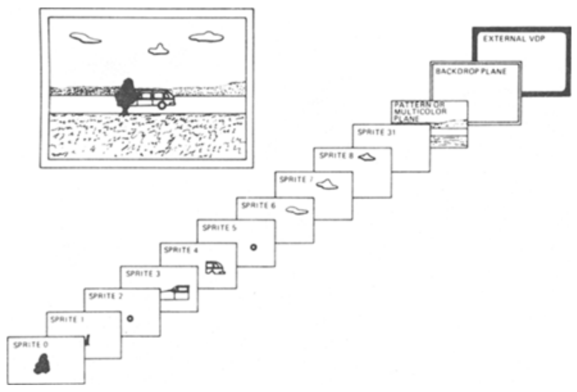

Figure 1. A computer-generated scene created through the combination of multiple display planes. Each video plane is assigned a priority of visibility as a function of its position in depth relative to the observer. [From *Texas Instruments TMS 9918A VDP Data Manual* (Texas Instruments, 1981), reprinted courtesy of Texas Instruments, Incorporated]

In sum, the TMS 9918A VDP incorporates all of the features of other VDPs, but its unique architecture enables the user to program smooth motion of certain graphics shapes (sprites) with multilevel pattern overlaying. These features make it possible for investigators interested in motion perception phenomena to program dynamic events with a modicum of programming skill.

GENERAL PRINCIPLES OF OPERATION FOR THE TMS 9918A VDP

The objective of this section is to provide a general overview of the procedure involved in programming the TMS 9918A VDP. [More specific information can be obtained from the *Texas Instruments TMS 9918A VDP Data Manual* (Texas Instruments, 1981).] There are two basic components of this task: (1) writing the data that define the patterns and their locations on the graphics screen to a buffer of dynamic memory (henceforth referred to as video random access memory–VRAM) attached directly to the VDP; and (2) initializing the eight registers of the VDP.

The data are organized into five discrete blocks in VRAM. Two define the sprite patterns, and the other three define the pattern appearing on the background plane. With regard to the sprites, one block contains the pattern definitions. They are arranged into subblocks of 8 bytes each and are bit-mapped such that each row of an 8 x 8 sprite corresponds to 1 byte of the block. Each bit assigned "1" causes the sprite to be defined at that point; each bit assigned "0" causes the sprite to be transparent at that point. A 16 x 16 sprite can be constructed from four 8-byte subblocks corresponding to four 8 x 8 sprites arranged as the quadrants of a square. (As many as 256 8 x 8 sprites or 64 16 x 16 sprites can be specified in the Pattern Definition Table at one time.) The second block specifies a Sprite Attribute Table containing the vertical and horizontal positions of the sprite, its video plane assignment corresponding to 1 of the 32 planes available, and its color. In order to move a sprite on the screen, it is necessary only to change the x,y coordinates in the Sprite Attribute Table; the color of the sprite can be changed in the same manner.

The pattern plane is defined by the three other blocks consisting of a Pattern Definition Table, a Pattern Name

Table, and a Pattern Color Table. These three tables provide a library of colored pattern blocks and their screen locations. Although these tables are similar to those created for displaying sprites, the logic governing how a background pattern is defined and displayed is somewhat more complex and will not be described further, since our primary emphasis is on the generation of sprites.

The eight registers of the VDP contain information concerning different modes of operation as well as the starting addresses for each of the tables stored in VRAM. For the purpose of displaying sprites, it should be noted that Register 5 contains the upper 7 bits of the 14-bit Sprite Attribute Table address and Register 6 contains the upper 3 bits of the 14-bit address of the Sprite Definition Table. The lower 4 bits of Register 7 contain the color code for the backdrop plane. Finally, Bits 6 and 7 of Register 1 specify sprite size ("0" selects 8 x 8 pixels; "1" selects 16 x 16 pixels) and magnitude ("0" selects 1 pixel per array element; "1" selects 2 x 2 pixels per array element), respectively.

Listing 1 presents a general-purpose machine language driver for the TMS 9918A VDP written in 6502 assembly language. It is designed so that separate routines can be called to initialize the registers, clear VRAM, and load specific blocks of data to VRAM. Note that each block must be loaded into that portion of VRAM specified by the corresponding VDP register address for that particular function, for example, sprite attributes, sprite pattern definitions, etc. This goal is accomplished by loading a 2-byte address setup value into the VDP at the beginning of the data-transfer routine that is equivalent to the address specified by the corresponding VDP register. In addition, Bit 6 of the second byte of the address setup value must be set high to indicate that the CPU is transferring data to the VRAM through the VDP. This transfer uses a 14-bit autoincrementing address register; thus, once the starting address is set up, data are transferred 1 byte at a time to the VRAM. In a similar fashion, data are written to each of the eight registers in the VDP by transferring the data in the first byte and selecting the register in the second. The chip convention requires that this second byte have the most significant bit (MSB) set high, the next 4 bits set low, and the lowest 3 bits set to correspond to the destination register number.

In our lab, we have used this driver for creating a number of different dynamic displays on an Apple II microcomputer. One particular application displays wheel-generated motion and illustrates especially well how to use the machine language driver in conjunction with a program written in a higher level language.

Listing 1
Machine Language Driver for Displaying Sprites

```
                  ************************************
                  *          SPRITE DRIVER           *
                  *    BY BENNETT I. BERTENTHAL      *
                  *          MARCH 2, 1984           *
                  ************************************

                  * This driver performs the following functions:
                  *   1. Initializes 8 registers of the Video
                  *      Display Processor (VDP)
                  *   2. Clears video RAM (VRAM)
                  *   3. Loads any number of sprites (0-32)
                  *      and their pattern definitions

                  * All blocks of data to be loaded into VRAM will
                  * have their starting and ending address loaded
                  * into 4 consecutive locations on Page zero.
                  *   Starting address:  LBYTE = $06, HBYTE = $07
                  *   Ending address:    LBYTE = $08, HBYTE = $09

                  * This program assumes that all blocks of data
                  * begin at a page boundary.

                  * This program also assumes that the sprite card
                  * is in slot 4.

                           ORG  $300
                           OBJ  $300

                  ****  DEFINE LABELS *****

                  SLOT     EQU  $40          ;SLOT # FOR SPRITE CARD
                  BYTE2    EQU  $1A          ;SET-UP ADDRESS FOR TRANSFERRING DATA TO VRAM
                  VREG     EQU  $C081+SLOT ;VDP REGISTER
                  VRAM     EQU  $C080+SLOT ;VDP RAM
                  REGADR   EQU  $380         ;STARTING ADDRESS OF  REGISTERS

                  ****  INITIALIZE REGISTERS  ****

0300: A0 80                LDY  #$80         ;SELECT REGISTER
0302: A2 00                LDX  #$00         ;INITIALIZE COUNTER
0304: BD 80 03    LOOP     LDA  REGADR,X     ;LOAD DATA FOR REGISTER
0307: 20 44 03             JSR  SREG         ;WRITE TO VDP
030A: C8                   INY               ;INCREMENT SELECTED REGISTER
030B: E8                   INX               ;INCREMENT SELECTED COUNTER
030C: E0 08                CPX  #$08         ;ALL REGISTERS LOADED?
030E: D0 F4                BNE  LOOP         ;IF NOT, BRANCH
0310: 60                   RTS

                  ****  CLEAR VRAM MEMORY  ****

0311: A4 1A                LDY  BYTE2        ;SET-UP ADDRESS - BYTE2
0313: A9 00                LDA  #$00         ;SET-UP ADDRESS - BYTE1
0315: 20 44 03             JSR  SREG         ;WRITE TO VDP
0318: A2 C0                LDX  #$C0         ;COUNTER - HIGH BYTE
031A: A0 00                LDY  #$00         ;COUNTER - LOW BYTE
031C: 8D C0 C0    COUNT    STA  VRAM         ;WRITE 0 TO VDP RAM
031F: C8                   INY               ;INCREMENT LOW COUNTER
0320: D0 FA                BNE  COUNT        ;IS LOW COUNTER FULL?
0322: E8                   INX               ;INCREMENT HIGH COUNTER
0323: D0 F7                BNE  COUNT        ;HIGH COUNTER FULL?
0325: 60                   RTS


                  ****  LOAD VRAM TABLES  ****

0326: A4 1A                LDY  BYTE2        ;SET-UP ADDRESS - BYTE2
0328: A9 00                LDA  #$00         ;SET-UP ADDRESS - BYTE1
032A: 20 44 03             JSR  SREG         ;WRITE TO VDP
032D: A0 00                LDY  #$00         ;INITIALIZE COUNTER
032F: B1 06       MORE     LDA  ($06),Y      ;LOAD DATA
0331: 8D C0 C0             STA  VRAM         ;WRITE TO VDP RAM
0334: C8                   INY               ;INCREMENT COUNTER
0335: D0 02                BNE  SKIP         ;FIRST PAGE FULL?
0337: E6 07                INC  $07          ;YES, INCREMENT HBYTE
0339: A6 07       SKIP     LDX  $07          ;LOAD HBYTE
033B: E4 09                CPX  $09          ;COMPARE TO HBYTE FOR END OF TABLE
033D: D0 F0                BNE  MORE         ;LAST PAGE?
033F: E4 08                CPX  $08          ;YES, COMPARE TO LBYTE FOR END OF TABLE
0341: D0 EC                BNE  MORE         ;LAST PAGE DONE?
0343: 60                   RTS

                  ****  WRITE TO VDP ****

0344: 8D C1 C0    SREG     STA  VREG         ;STORE BYTE1
0347: 8C C1 C0             STY  VREG         ;STORE BYTE2
034A: 60                   RTS


--End assembly--

75 bytes

Errors: 0
```

AN ILLUSTRATIVE EXAMPLE: WHEEL-GENERATED MOTION

The study of wheel-generated motion has had a long and venerable history in the visual perception literature (Proffitt & Cutting, 1979). In essence, this event involves the presentation of two or more points of light moving as if placed on various locations of a rotating invisible wheel translating across a surface. Although observers typically report seeing the lights rotate around the centroid (center of rotation for any system of lights), an analysis of the motions of the individual lights reveals that each moves along a path describing a cycloid or prolate cycloid (see Figure 2). The one exception is the point at the center of the wheel that moves along a linear path (Proffitt & Cutting, 1979). For example, the absolute motions of two lights located on the rim of a wheel 180 deg apart from each other both trace identical cycloidal paths corresponding to the same sinusoidal function differing only in phase. It follows therefore that programming wheel-generated motion on a computer involves displaying a series of lights that all undergo sinusoidal motions along a common path with the motion of each light out of phase with the others.

Our approach to creating wheel-generated motion was to keep the task as simple as possible by programming all initial procedures in Applesoft BASIC (see Listing 2). These procedures include loading the machine language driver and Sprite Pattern Table, initializing variables, and computing and storing in arrays the x and y coordinates for each sprite prior to the time that it is displayed on the screen. (An even more efficient procedure is to poke these values directly into RAM.) Thus, the processing demands during the display portion of the program are reduced to simply reading the next pair of x,y values from the array and writing them via the machine language driver into the appropriate location of the Sprite Attribute Table in VRAM. In addition, the program is compiled (Microsoft Tasc Compiler) not only to further reduce processing time during the display portion of the program, but also to reduce processing time during the earlier portion when the x and y coordinates are computed.

This program allows the user to display between one

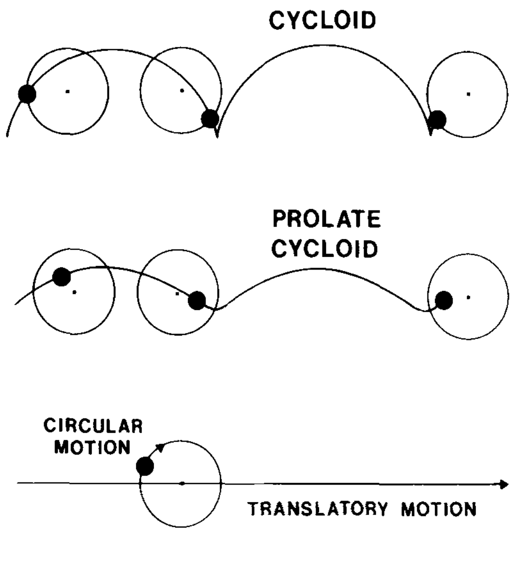

Figure 2. In the top panel is shown the motion of two points on a rolling wheel, with the center represented as a small dot and a perimeter point as a large dot. The middle panel likewise presents the motion of two points on a rolling wheel, one being the center and the other a point within the perimeter. The lower panel shows the circular and translatory components of motion. (From Proffitt & Cutting, 1979)

Listing 2
Demonstration Program for Displaying Wheel-Generated Motions

```
10   REM  ********************************
20   REM  *   WHEEL-GENERATED MOTIONS    *
30   REM  *   BY BENNETT I. BERTENTHAL   *
40   REM  *      & STEVEN J. KRAMER      *
50   REM  *        MARCH 2, 1984         *
60   REM  ********************************
70   REM
80   HOME : VTAB 7: HTAB 8: INVERSE : PRINT  SPC( 23)
90   VTAB 8: HTAB 8: PRINT "WHEEL-GENERATED MOTIONS"
100  VTAB 9: HTAB 8: PRINT  SPC( 23): NORMAL
110  VTAB 12: HTAB 5: PRINT "THIS PROGRAM DISPLAYS AS MANY AS"
120  VTAB 13: HTAB 5: PRINT "6 LIGHTS ROTATING AND TRANSLATING"
130  VTAB 14: HTAB 5: PRINT "AROUND THE CENTROID OF A WHEEL"
140  VTAB 23: HTAB 8: INVERSE : PRINT "PRESS ANY KEY TO CONTINUE": NORMAL : GET
     Z$: PRINT Z: HOME
150  REM
160  REM  ********************************
170  REM  *  INITIALIZE VARIABLES AND    *
180  REM  * LOAD MACHINE LANGUAGE PRGMS  *
190  REM  ********************************
200  REM
210  DIM X(300,6),Y(300,6),COL(6),N(6)
220  VTAB 10: HTAB 8: INVERSE : PRINT "LOADING SPRITE DRIVER": VTAB 12: HTAB 9:
     PRINT " AND PATTERN TABLE": NORMAL
230  PRINT  CHR$ (4);"BLOAD SPRITE PATTERN TABLE, A$8100": PRINT  CHR$ (4);
     "BLOAD SPRITE DRIVER, A$300": HOME
240  VTAB 10: HTAB 9: INVERSE : PRINT "  PLEASE BE PATIENT   ": VTAB 11: HTAB 9:
     PRINT "INITIALIZING VARIABLES": NORMAL
250  REM
260  REM  DEFINE CONSTANTS
270  PI = 3.14159:DEG = 57.2956
280  REM
290  REM   DEFINE VARIABLE ADDRESSES
300  BYTE2 = 26: REM  MEMORY LOCATION FOR VRAM ADDRESS SET-UP VALUE
310  R0 = 896:R1 = R0 + 1:R2 = R1 + 1:R3 = R2 + 1:R4 = R3 + 1:R5 = R4 + 1:
     R6 = R5 + 1:R7 = R6 + 1: REM     REGISTER LOCATIONS
320  SPT = 32768:PT = SPT + 256: REM  STARTING LOCATIONS FOR SPRITE ATTRIBUTE
     AND PATTERN TABLES
330  SL = 6:SH = 7:EL = 8:EH = 9: REM  POINTERS FOR BEGINNING AND END OF BUFFERS
340  IR = 768:CM = 785:TD = 806: REM    STARTING LOCATIONS FOR MACHINE LANGUAGE
     ROUTINES
350  REM
360  REM  DEFINE FUNCTIONS
370  DEF  FN LO(X) =  INT (X -  INT (X / 256) * 256)
380  DEF  FN HI(X) =  INT (X / 256)
390  REM
400  REM  ********************************
410  REM  *   USER-DEFINED VARIABLES    *
420  REM  ********************************
430  HOME : VTAB 3: HTAB 10: INVERSE : PRINT "USER-DEFINED VARIABLES": NORMAL
440  VTAB 5: HTAB 5: PRINT "ENTER X(5-30), Y(10-150) "

450  VTAB 6: HTAB 5: INPUT "FOR LIGHT #1: ";CX$,CY$:CX =  VAL (CX$):
     CY =  VAL (CY$): IF CX < 5 OR CX > 30 OR CY < 10 OR CY > 150 THEN
     GOSUB 1030: GOTO 450
460  VTAB 8: HTAB 5: INPUT "WHEEL RADIUS (5-30) = ";R$:R =  VAL (R$):
     IF R < 5 OR R > 30 THEN  GOSUB 1030: GOTO 460
470  VTAB 10: HTAB 5: INPUT "ROTATION INCREMENT (2-30) = ";RI$:RI =  VAL (RI$):
     IF RI < 2 OR RI > 30 THEN  GOSUB 1030: GOTO 470
480  VTAB 12: HTAB 5: INPUT "TRANSLATION SPEED (0-5) = ";TS$:TS =  VAL (TS$):
     IF TS < 0 OR TS > 5 THEN  GOSUB 1030: GOTO 480
490  VTAB 14: HTAB 5: PRINT "CORRECTION FACTOR FOR"
500  VTAB 15: HTAB 5: INPUT "SCREEN (0.65-1.00) = ";CF$:CF =  VAL (CF$):
     IF CF < .65 OR CF > 1.0 THEN  GOSUB 1030: GOTO 500
510  FOR I = 1 TO 6
520  VTAB 17 + I: HTAB 5: PRINT "LIGHT # ";I;: INPUT "  COLOR (0-15) = ";COL$:
     COL(I) =  VAL (COL$): IF COL(I) < 0 OR COL(I) > 15 THEN  GOSUB 1030: GOTO 520
530  POKE SPT + 3 + ((I - 1) * 4),COL(I): REM   POKE SPRITE COLOR VALUE
540  POKE SPT + 2 + ((I - 1) * 4),I - 1: REM    POKE SPRITE NAME
550  NEXT I
560  REM  ********************************
570  REM  *     INITIALIZE REGISTERS    *
580  REM  ********************************
590  POKE R0,2: POKE R1,192: POKE R2,7: POKE R3,27: POKE R4,255: POKE R5,62:
     POKE R6,3: POKE R7,1
600  CALL IR: REM   CALL MACHINE LANGUAGE ROUTINE FOR TRANSFERRING VALUES INTO
     VDP REGISTERS
610  REM
620  REM CLEAR VIDEO RAM
630  CALL CM: REM   MACHINE LANGUAGE ROUTINE FOR CLEARING VRAM
640  A = PT: GOSUB 1130:A = PT + 48: GOSUB 1140
650  POKE BYTE2,88: CALL TD: REM  MACHINE LANGUAGE ROUTINE FOR TRANSFERRING SPRITE
     PATTERN TABLE INTO VRAM
660  REM  ********************************
670  REM  *  COMPUTE X & Y COORDINATES  *
680  REM  *        FOR EACH LIGHT        *
690  REM  ********************************
700  HOME : VTAB 10: INVERSE : PRINT "COMPUTING X & Y COORDINATES": NORMAL
710  REM
720  REM  INITIALIZE COUNTERS
730  NC = 0: REM   INITIALIZE COUNTER FOR LOOP
740  TR = 0: REM  INITIALIZE TRANSLATION DISTANCE
750  REM
760  REM  COMPUTE LOCATION OF LIGHTS AROUND WHEEL (IN RADIANS)
770  N(1) = N(1) + RI / DEG:N(2) = N(1) + PI / 2:N(3) = N(1) + PI
780  N(4) = N(3) + PI / 2:N(5) = N(1) + PI / 4:N(6) = N(1) + PI / 8
790  FOR I = 1 TO 6: IF I = 6 THEN RR = R:R = R * .6
800  X(NC,I) = CX + R *  SIN (N(I)) + TR
810  Y(NC,I) = (CY - R *  COS (N(I))) * CF
820  IF X(NC,I) > 255 THEN X(NC,I) = 255: GOTO 860
830  NEXT I
840  PRINT ".";
850  R = RR:NC = NC + 1:TR = TR + TS: GOTO 770
860  REM  ********************************
870  REM  *       DISPLAY LIGHTS         *
880  REM  ********************************
890  HOME : VTAB 10: HTAB 9: PRINT "PRESS SPACE BAR TO BEGIN": VTAB 11:
     HTAB 12: PRINT "DISPLAYING LIGHTS": GET A$: POKE  - 16368,0:
     IF A$ <  >  CHR$ (32) THEN 890
900  VTAB 16: HTAB 10: INVERSE : PRINT "PRESS ANY KEY TO STOP": NORMAL
```

```
910  REM
920  REM  TRANSFER X & Y COORDINATE VALUES TO SPRITE ATTRIBUTE TABLE
930  POKE BYTE2,95:A = SPT: GOSUB 1130:A = SPT + 24: GOSUB 1140
940  FOR I = 1 TO NC
950  FOR J = 1 TO 6
960  POKE SPT + ((J - 1) * 4),Y(I,J): POKE SPT + 1 + ((J - 1) * 4),X(I,J)
970  NEXT J
980  CALL TD
990  IF  PEEK ( - 16384) > 128 THEN 1020
1000  NEXT I
1010  GOTO 940
1020  END
1030  REM  ******************************
1040  REM  *  SUBROUTINE FOR PRINTING  *
1050  REM  *   INPUT ERROR TO SCREEN   *
1060  REM  ******************************
1070  CV =  PEEK (37): VTAB CV: HTAB 5: CALL  - 868: INVERSE : PRINT "OUT OF
      RANGE VALUE": FOR II = 1 TO 350: NEXT II: NORMAL
1080  VTAB CV: HTAB 5: CALL  - 868: RETURN
1090  REM  ******************************
1100  REM  *   SUBROUTINE FOR POKING   *
1110  REM  *STARTING & ENDING LOCATIONS*
1120  REM  ******************************
1130  POKE SL, FN LO(A): POKE SH, FN HI(A): RETURN
1140  POKE EL, FN LO(A): POKE EH, FN HI(A): RETURN
```

and six sprites rotating around the center of an invisible wheel translating from left to right across the screen. The sprites are designed to appear as small circles of light moving on a dark background. Each sprite contains a separate pattern definition stored in the Sprite Pattern Table. (A binary dump of this table is presented in Listing 3.) The initial locations of the sprites around the circle are presented in Figure 3. These locations are presented in radians rather than degrees because the trigonometric functions in Applesoft expect the argument to correspond to radians.

Some of the parameters of the display must be specified by the user in Lines 300-430. These user-defined parameters include: (1) initial location for sprite #1 (20,85)[2]; (2) wheel radius (measured in pixels displayed on the screen) (20); (3) rotation increment (5)–this value corresponds to the number of degrees incremented around the circle each time the positions of the sprites are updated; (4) translation speed (1.7)–this value corresponds to the number of pixels incremented on the horizontal axis each time the location of the sprites is updated; (5) correction factor for screen (.80)–this value is designed to correct for screens for which the aspect ratio between horizontal and vertical pixels is not equal to 1.00; (6) color for each of the six sprites–the value 15 corresponds to white, and the value 0 corresponds to transparent. It is thus possible to control the number of sprites to be displayed by setting the color of unwanted sprites to 0.

Listing 3
Binary Dump of Sprite Pattern Table

```
*8100.812F

8100- 00 30 78 78 30 00 00 00
8108- 00 30 78 78 30 00 00 00
8110- 00 30 78 78 30 00 00 00
8118- 00 30 78 78 30 00 00 00
8120- 00 30 78 78 30 00 00 00
8128- 00 30 78 78 30 00 00 00
```

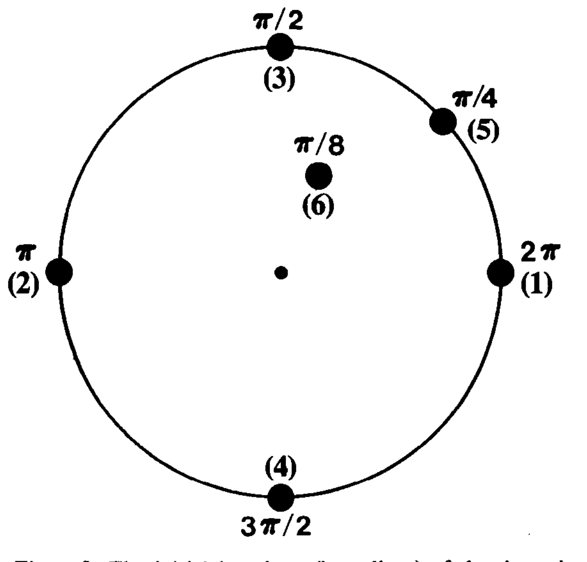

Figure 3. The initial locations (in radians) of the six sprites around the wheel.

ADDITIONAL APPLICATIONS IN THE CREATION OF DYNAMIC DISPLAYS

Although the above program is a good illustration of how sprites can be used in dynamic displays, it by no means exhausts the possible applications or makes use of all of the special features provided by the TMS 9918A VDP. In this last section, we will illustrate some of the other features available to the user by discussing the creation of two additional displays: segregation of two randomly textured surfaces through kinetic occlusion, and specification of a human form through biomechanical motions.

Kinetic Occlusion of Randomly Textured Surfaces

Two randomly textured surfaces appear as a flat continuous surface as long as both remain stationary. Yet, as soon as one begins to move relative to the other, there occurs an accretion of texture at the trailing edge of the nearer surface and a deletion of texture at the leading edge. Observers report seeing one surface going behind another at an occluding edge and use this information for specifying which of the two surfaces is nearer (Gibson et al., 1968). Kaplan (1969) was the first to test this phenomenon empirically by creating a movie film of two moving randomly textured surfaces. This approach must have been painstakingly slow, since each frame had to be filmed separately. In contrast, we created this demonstration in a relatively short period of time using both sprites and the pattern plane of the TMS 9918A VDP.

The first step was to create a random texture pattern using a dart-throwing technique to determine which pixels should be colored (the probability of lighting any pixel was .3). This pattern was programmed in Apple II high-resolution graphics and then converted to a pattern that could be displayed by the TMS 9918A VDP using a commercially available conversion program (E-Z Color Graphics Editor, by Micromint). The nearer occluding surface was created from sprites designed with the same dart-throwing algorithm used for creating the larger pattern plane. Note that the size of the occluding surface need not be constrained by the size of a single sprite, since a moving unitary shape can be composed of more than one sprite. For example, the car displayed in Figure 1 was composed of two sprites that occupied different screen locations and were assigned to two successive video planes. For the purpose of this demonstration, we arranged four sprites together in such a way that each sprite represented one quadrant of an array twice as large as a single sprite. When the sprites are programmed to move back and forth over the pattern plane, they produce an accretion and deletion of texture of the more distant surface because the sprites automatically occlude the texture on the more distant pattern plane.

Although the above procedure was quite successful in creating a demonstration of the kinetic disruption of optical texture, a small caveat is in order. Recall that each sprite is assigned only one color; thus, all pixels not corresponding to the pattern remain transparent. Accordingly, this particular demonstration of surface segregation through texture occlusion differs from Kaplan's (1969) original demonstration in one unusual way. Whereas Kaplan's demonstration involved two opaque surfaces, this new version created with sprites depicts a moving transparent surface occluding a more distant opaque surface. Phenomenally, the nearer surface appears to be like a thin pane of glass with a texture pattern randomly distributed over its area. A simple procedure for modifying the present demonstration to depict an opaque moving surface is to add four additional sprites directly behind the four composing the moving occluding surface. As long as these sprites are programmed to represent a solid pattern sharing the same color as the background, and all lie in video planes that will be occluded by the four sprites defining the texture of the moving surface, then the nearer moving surface will appear opaque.

In sum, the creation of this demonstration highlights two points about using sprites in the generation of dynamic displays: (1) The creation of kinetic occlusion is reduced from an extremely complex programming task to one that is relatively simple when the TMS 9918A VDP is used; and (2) the size of any one moving shape is not limited by the size of a sprite, but rather is defined by the number of contiguously arranged sprites that move together.

Biomechanical Motions

Johansson (1973) showed that adults can almost immediately recognize the form of a person walking when they view a two-dimensional projection of 10 to 12 point-lights attached to the head and major joints of an unseen walker (see Figure 4A). This same event has more recently been simulated on a computer by Cutting (1978). In essence, Cutting has written a program that displays points of lights mimicking the movements of a human walker. These lights appear to be mounted on the major joints (near shoulder and hip, and both wrists, elbows, knees, and ankles) and head of a person walking along a path normal to the observer's line of sight. There exists, however, one significant difference between this computer-synthesized display and those created using real people. In versions of this event involving the filming of real people, there appears a gradual deletion and accretion of the more distant point-lights as they are hidden and unhidden by the invisible form. Specifically, the more distant wrist, elbow, knee, and ankle, as well as the hip light, undergo some degree of kinetic occlusion twice during each gait cycle. In contrast, the computer simulation has each of the relevant point-lights disappear and appear all at once as if the point-lights were blinking on and off. When one considers how much complexity is added to the programming task by trying to achieve gradual accretion and deletion of the point-lights, it is not surprising that this effect was omitted.

An implementation of the point-light walker program with the TMS 9918A VDP overcomes this programming limitation by using sprites to represent both the point-lights and the invisible occluding forms. In essence, we used the basic logic of Cutting's (1978) FORTRAN program, but rewrote it in Applesoft BASIC and also modified it to write the x,y coordinates to the sprite driver rather than to an oscilloscope driver. In this sprite-oriented version, Sprites 0-6 and 11-14 are used to

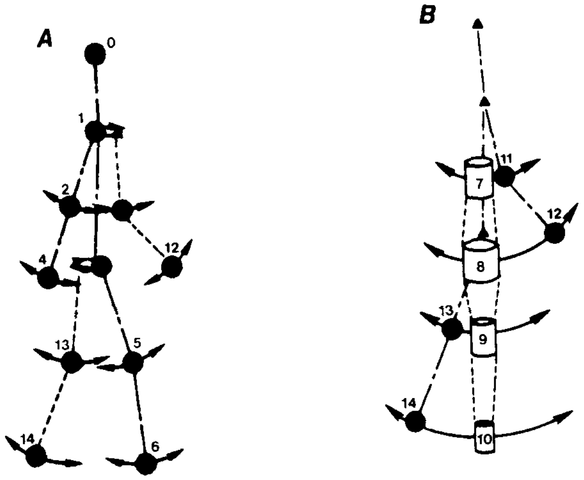

Figure 4. Depicted in A is the array of 11 point-lights attached to the head and joints of a walking person: The head and near side of the body are numbered 0 through 6, and the more distant joints are numbered 11 through 14. These numbers correspond to the priority of visibility possessed by each sprite used to display that particular point-light. Depicted in B is the shape and relative location of those sprites (7 through 10) used to occlude the far joints (11 through 14) of the walker as they pass behind the invisible form of the body.

represent the point-lights in the display and are arranged so that the higher numbered (lower priority, or more distant) sprites correspond to the point-lights depicting the more distant joints (see Figure 4A). Sandwiched between the more distant point-lights and those appearing closer are four additional sprites (Sprites 7-10) shaped to correspond to the human form at those relative positions (see Figure 4B). These four sprites are not visible to the observer because they are of the same color as the background; still, they serve to gradually hide and unhide the sprites corresponding to the far joints of the walker, since these background-colored sprites have higher priorities of visibility.

There is one additional point-light, corresponding to the hip, that should also undergo gradual accretion and deletion, but does not in the present scheme. The reason for this omission highlights the only significant limitation we have found in using the TMS 9918A VDP. As currently designed, this VDP can display only four sprites on any one horizontal line. If more than four are programmed to appear on the same line, then the four highest priority (or closest) sprites on the line are displayed normally and the fifth and subsequent sprites are not displayed at all. In order to occlude the hip light as the implicit arm of the walker passes in front of it, an additional sprite would have to be sandwiched between the near wrist and hip. This manipulation is not possible, however, because there are already four sprites overlapping in the same location; these sprites correspond to the hip, the two wrists, and the invisible occluder sandwiched between the hip and the more distant wrist light. Although there are procedures for getting around this limitation (such as computing the period during the gait cycle when the hip light should be occluded and then changing the priority value of the occluding sprite for that portion of the gait cycle), it must be recognized that these procedures can be implemented only by adding to the complexity of the programming task.

In spite of the one limitation described above, the ease of depicting the gradual hiding and unhiding of the point-lights remains an important advantage in using sprites for programming point-light displays of biomechanical motions. Furthermore, these displays appear almost as smooth and continuous as those created by Cutting (1978) using an HP 1350A display system driven by an HP 1000L computer; thus, they provide a further endorsement for the effectiveness of sprite-oriented graphics in the creation of moving displays.

CONCLUDING REMARKS

As recently as 3 years ago, we were convinced that it was impractical to create dynamic displays on a microcomputer. Since that time, we have begun to create moving displays with a microcomputer using sprites and other features provided by the TMS 9918A VDP. Many of our applications have involved moving point-light displays, for which sprites are especially well suited. Still other types of dynamic displays are also possible, as demonstrated by our discussion of the kinetic disruption of optical texture. Most importantly, we are finding that many of our moving displays are comparable to those created with much more expensive minicomputers and oscilloscopes.

REFERENCES

Cavanagh, P., & Anstis, S. M. (1980). Visual psychophysics on the Apple II: Getting started. *Behavior Research Methods & Instrumentation*, **12**, 614-626.

Cutting, J. E. (1978). A program to generate synthetic walkers as dynamic point-light displays. *Behavior Research Methods & Instrumentation*, **10**, 91-94.

Gibson, J. J., Kaplan, G. A., Reynolds, H. N., & Wheeler, K. (1968). The change from visible to invisible: A study of optical transition. *Perception & Psychophysics*, **5**, 113-116.

Jochumson, C. (1983). Arcade graphics techniques. *Call A.P.P.L.E.*, **6**, 9-14.

Johansson, G. (1973). Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, **14**, 201-211.

Johansson, G., von Hofsten, C., & Jansson, G. (1980). Event perception. *Annual Review of Psychology*, **31**, 27-63.

Kaplan, G. A. (1969). Kinetic disruption of optical texture: The perception of depth at an edge. *Perception & Psychophysics*, **6**, 193-198.

Proffitt, D. R., & Cutting, J. E. (1979). Perceiving the centroid of configurations on a rolling wheel. *Perception & Psychophysics*, **25**, 389-398.

Texas Instruments, Inc. (1981). *Texas Instruments TMS 9918A video display processor data manual*. Houston, TX: Author.

NOTES

1. This chip is designed to interface with a host of microprocessors via an 8-bit bidirectional data bus and three control lines. Its output is a composite color video signal defined by the contents of dynamic VRAM (ranging between 4K and 16K) attached directly to the VDP. The VRAM is automatically refreshed by the VDP and needs no direct connections to the host computer. Video output is asynchronous with other functions of the VDP, and thus the host processor and the VDP can communicate at any time without interfering with the video output.

2. A prototypical display is created with the values enclosed in parentheses following the listing of each user-defined variable.

(Manuscript received March 31, 1984;
revision accepted for publication June 5, 1984.)