SESSION IV COMPUTER RESEARCH TOOLS

Paper Session

3-D graphics animation program for the Apple II

BENNETT I. BERTENTHAL, DENNIS R. PROFFITT, and STOWE E. KELLER
University of Virginia, Charlottesville, Virginia

A graphics design program for creating point-light displays of transforming 3-D objects is presented. This program was written for an Apple II microcomputer interfaced to a Texas Instruments TMS 9918A video display processor. In contrast to other 3-D design programs, it uses individual point-lights undergoing circular trajectories as a design primitive. An editor enables the user to enter and edit specific motion parameters for defining the parallel projection of as many as 16 point-lights on the screen. These parameters are then used to calculate and store in an animation list the screen positions of each point-light for each frame. Number of frames to be displayed and display rate are user defined.

This paper describes a 3-D graphics design program for creating point-light displays of transforming three-dimensional objects. The program can be implemented on an Apple II microcomputer and minimizes the need for knowing the mathematics underlying the parallel projection of a moving object onto a two-dimensional surface. Thus, this approach to computer animation is both cost effective and simple to implement.

BACKGROUND

A fundamental question to researchers in visual perception and computer vision concerns how a visual system extracts a coherent three-dimensional structure from a transforming two-dimensional image. In order to determine whether or not such information can be extracted from the transformations alone, point-light displays are often used in studies involving projections of moving rigid or jointed objects. For example, one frequently used event involves point-lights moving as though attached to the rim of an invisible wheel rotating along the picture plane or appearing to rotate around an axis tilted in depth (Proffitt & Cutting, 1979). The mathematical tractability of this display makes it well suited for learning about the extraction of structure from a continuously changing image.

An even more interesting display is that of an array of

point-lights moving as though attached to the head and major joints of a person who is walking (Johansson, 1973). In contrast to point-light displays depicting transforming rigid objects, this display presents an even more complex problem to the observer, since the projected distances between almost all of the joints are continuously changing. Johansson created his displays by placing small lights on a person who was then filmed while walking in the dark (see Figure 1A). Although the film consisted of only 10 to 12 points of light moving across a dark background, it was quickly and unequivocally recognized as a walking person; static frames were not recognized as anything familiar. These displays were later created by Cutting (1978) using computer synthesis. An important advantage of Cutting's approach is that individual parameters can be precisely manipulated. Thus, it is possible to test, for example, whether or not the spatial locations of the point-lights are important by adding random values to the x,y coordinates (Figure 1B).

In our own research, we have devised many ways to manipulate the base algorithm of the point-light walker display to learn more about the extraction of a unique three-dimensional interpretation from this display (Bertenthal, Proffitt, & Spetner, 1984; Proffitt, Bertenthal, & Roberts, 1984). Still, there are numerous limitations to studying a problem about basic processes using the projection of a very familiar object, such as the human form, since it is difficult to evaluate the contribution of stored knowledge. The realization of these limitations prompted us to shift our emphasis to studying moving point-light projections of unfamiliar jointed objects.

The creation of new point-light displays represented a significant challenge, since we knew neither the stimulus parameters nor the processing constraints that would prove

The development of this program was supported by NICHD Grant HD-16195. We wish to thank Steven Kramer 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. Hard copies of the program are available from the same address. Please enclose \$5.00 to cover the cost of a disk and postage.

Figure 1. Panel A shows a display of 11 point-lights moving as though attached to the head and major joints of a walking person. The motion vectors drawn through each point represent the perceived relative motions of the display. Panel B depicts a scrambled walker display that is identical to A except that the relative locations of the point-lights are scrambled. Correspondingly numbered points in the two panels have the same absolute motions.

most informative. We thus recognized that the most effective approach for studying the extraction of structure from motion was to design a flexible program not centered around any particular base algorithm. The result was the development of a computer-graphics design program that allows for the creation of point-light parallel projections of virtually any transforming rigid, jointed, or elastic object.

UNDERLYING LOGIC FOR DESIGN PROGRAM

Currently, most design programs are concerned with geometric modeling of individual rigid solids (Requicha, 1980). These programs reflect a primary interest in the adequacy of various representations for closed volumes in the task of designing, analyzing, and possibly manufacturing specified objects. Representations such as edge-face lists (Braid, 1978), spatial enumeration (Herman & Webster, 1980), surface patches (Barnhill & Riesenfeld, 1974), and constructive solid geometry (Requicha, 1980) have been used in attempts to completely describe the surface and thereby the volume of single objects.

Our system is not concerned with the surface details of rigid objects; rather, its primary goal is the specifica-

tion of multicomponent motions in point-light displays. In contrast to primitive solids, such as cylinders, cubes, and prisms, used in constructive solid geometry, our design program uses point-lights that can independently rotate and translate in any depth plane. The fundamental primitive is a point-light that moves in a circular path at a uniform velocity. By default, it rotates in a counterclockwise direction from the 3 o'clock position in the picture plane. The programming of a circular motion rotating in the picture plane is specified by the following three parameters (Restle, 1979):

- (1) Amplitude: the radius of rotation (r).
- (2) Wavelength: the number of cycles per second (360° represents one complete cycle).
- (3) Phase: the angular location (θ) of a point relative to 0° .

The formulas for specifying the trajectory of a point (x, y) on a circle (with its center assumed to lie at the origin $0, 0$ of the display screen) are given below:

$$x = r * \cos(\theta) \quad (1)$$

$$y = r * \sin(\theta). \quad (2)$$

The projection of a circle moving in depth relative to

the picture plane is specified by an elliptical path, which requires information about two additional parameters (Restle, 1979).

(4) Tilt: the angular rotation of the plane, on which occurs the circular trajectory, from the picture plane about the vertical axis. (Example—A point moving in a circle with a 90° tilt will project an image of a point moving up and down along a vertical line. Tilts other than 0°, 90°, 180°, and 270° will produce elliptical motions with the major axis aligned vertically.)

In essence, tilt is programmed into the circular trajectory of a point by producing a compression along the x-axis. The formulas used for producing a rotating point (tx,ty) at a specified tilt (τ) on the screen are given below:

$$tx = r * [\cos(\tau) * \cos(\theta)] \quad (3)$$

$$ty = r * \sin(\theta). \quad (4)$$

(5) Orientation: the angular rotation of the major axis in the picture plane. (Example—A point moving with a tilt of 90° and a 90° orientation will project an image of a point moving back and forth along a horizontal line. In general, orientation defines the alignment of the major axis of an ellipse relative to the vertical axis. For circles, orientation and phase are interchangeable.)

The formulas for specifying a point (ox,oy) undergoing a circular trajectory at a given orientation (β) and a given tilt (τ) are presented below:

$$ox = tx * \cos(\beta) - ty * \sin(\beta) \quad (5)$$

$$oy = tx * \sin(\beta) + ty * \cos(\beta) \quad (6)$$

where tx and ty are specified by Equations 3 and 4, respectively.

Taken together, these five parameters completely specify the parallel projection of a point-light rotating in any depth plane. By incorporating the calculations underlying these parameters within our Design Program, the programmer is freed from directly calculating the trajectories of the point-lights. In fact, the programmer need only enter information about the preceding five parameters (as well as a few others); the actual calculations needed for computing the trajectory of the point-lights remain invisible to the programmer.

GENERAL CHARACTERISTICS OF DESIGN PROGRAM

Sprite-Oriented Graphics

The Design Program was specifically written for an Apple II microcomputer interfaced to a Texas Instruments TMS 9918A Video Display Processor (VDP). Although smooth and continuous animation is a feature usually reserved for more expensive graphics display terminals or oscilloscopes driven by minicomputers, the TMS 9918A VDP produces moving point-light displays that are the

equivalent of much more powerful machines (Bertenthal & Kramer, 1984). The key to the success of this chip is a unique graphics architecture for displaying and moving a specialized graphic shape, referred to as a sprite.

In the current implementation, a sprite is a colored block shape corresponding to an 8 × 8 pixel (picture element) array. Its visible form depends upon which pixels are programmed to appear as a specific color and which are programmed to remain transparent. Sprites can be drawn on a video plane by specifying a single pair of x,y coordinates corresponding to the upper left-hand corner of the shape; full-screen resolution is 256 pixels horizontally and 192 pixels vertically. The movement of a sprite is accomplished simply by changing the x,y coordinates to some new value. It is thus possible to produce smoothly moving patterns at a resolution of 1 pixel.

Sprites also eliminate many of the programming complexities encountered when shapes pass in front of or behind each other on the screen. Each sprite is assigned a priority of visibility as a function of its position in depth relative to the observer; these values range between 0 and 15, since as many as 16 sprites can be displayed on the screen at the same time. Any time portions of two or more sprites occupy the same location on the screen, the portions of those sprites with the lower priorities (i.e., higher numbers) will be hidden by the portion of the sprite with the higher priority (i.e., lower number). As such, it is possible to program both static and kinetic forms of occlusion without any need for complex hidden surface algorithms.

In short, the TMS 9918A VDP incorporates all of the features found in other microcomputer VDPs, but its unique graphics architecture enables the programmer to produce smooth and continuous movement of block shapes that automatically occlude and disocclude one another. More specific details about the functioning and programming of this chip were presented in Bertenthal and Kramer (1984).

Overview of the Software

The complete Design Program requires 64K bytes of memory and can be executed on an Apple II PLUS, Apple IIe, or Franklin Ace 1000. After the Design disk is booted and DOS is loaded into memory, a start-up program, *HELLO*, automatically executes a public-domain machine language program (DOS MOVER, by Cornelis Bongers) for relocating DOS to a section of the 16K RAM card, corresponding to the highest portion of memory. This procedure maximizes the available program space in the lowest portion of memory. Next, a text file, *SETUP-CM*, containing a sequence of DOS commands is executed in order to load the main program, *CIRC.MOT*, written in Applesoft BASIC, and four machine language programs into memory. Presented below is a brief description of each of these programs.

CIRC.MOT is the main program and occupies 11.3K bytes of RAM. Its principal function is to interact with

Table 1
Ampersand Commands for Programming Sprites

Command	Action
1. &AT <slot #>	Sets the slot # for the TMS 9918A VDP interface.
2. &CLEAR	Erases the VDP RAM containing the data displayed on the screen.
3. &DIM <0 or 1>	Sets sprite size: 0 = 8 × 8 element array; 1 = 16 × 16 element array.
4. &SCALE = <0 or 1>	Sets sprite magnification: 0 = 1 pixel per array element; 1 = 2 × 2 pixels per array element.
5. &COLOR = <sprite #>, <color>	Assigns a color code (0-15) to the specified sprite number.
6. &DEF <sprite #>, <0 or 1>	Defines specified sprite number as either point-light or rectangle.
7. &LOAD	Loads the VDP registers and VDP RAM with the appropriate data.
8. &NEW	Sets the animation list pointer to the beginning of the list.
9. &STORE <sprite #> <x,y coord>	Stores the specified sprite number and its screen coordinates into the animation list.
10. &NEXT	Stores an end of frame marker into the animation list and advances to next frame.
11. &END	Stores an end of animation list marker at the end of the list.
12. &SPEED = <delay>	Sets a delay value for controlling the presentation rate of the animation list.
13. &RUN	Runs the animation sequence (display) stored in RAM.
14. &STEP <# of frames>	Displays only the specified number of frames from the animation list.
15. &SCRN <color>	Sets the screen backdrop color to the specified color code (0-15).
16. &PDL <0 or 1>	Specifies whether the presentation rate of the animation list is controlled by an argument from the &SPEED command (0) or by a variable value from the game paddle (1).
17. &LEN <var. 1>, <var. 2>	Returns the address and length in RAM of the animation list to variables 1 and 2, respectively.

the user for inputting and displaying parameters governing the trajectories of the point-lights. As the main program, it is also responsible for calling the machine language programs and providing error trapping and disk access. *AMPER.SPRITE* is a collection of machine language subroutines (1391 bytes in length) for displaying sprites with the Texas Instruments TMS 9918A VDP. Each subroutine, such as positioning a sprite or changing its color, is written as part of an ampersand library. Any function included in this library can be executed directly from Applesoft BASIC by using the appropriate ampersand command ("&" followed by a mnemonic for the subroutine and 0-2 numeric arguments). A complete listing of these commands is contained in Table 1. *SINETBL* is a 4096-byte data lookup table representing 1,024 4-byte floating-point sine values in the range of 0° to 90°. This table was included because the calculation of sine and cosine functions is extremely slow on a 6502 microprocessor. Calculation times were improved by a factor of 2 to 3 after the incorporation of this lookup table. *SINETBL.LOAD* is a short machine language program for relocating the sine table to a position in the upper 16K of RAM, accessible only through memory bank-switching. *SAN->6502* is the last machine language program (5870 bytes in length) and consists of five subroutines. The first three are designed to parse a string of characters representing an expression in standard algebraic notation (S.A.N.), compile them into a machine language object code, and numerically evaluate them. The fourth subroutine performs the calculations necessary for positioning the sprite at the correct location on every frame. Finally, the fifth subroutine controls character input and editing during parameter entry for the sprites. All of these

subroutines are executed from the main program through CALL commands.

RUNNING THE DESIGN PROGRAM

The principal functions of the Design Program include specifying the trajectories of the sprites, calculating their screen positions for each frame, storing the values for each frame in an animation list, and sequentially displaying these frames at various animation speeds. The Design Program is divided into the following seven options: (1) Enter/Edit, (2) Calculate Animation List, (3) Run Animation List, (4) Disk Access, (5) Print Sprite Descriptions, (6) Specify Configuration, (7) Exit to BASIC. Each option is directly available from a screen menu. A user can return to this menu at any time by pressing the Escape key. The operation of each option is described below.

Enter/Edit

This option enables the user to either input or edit the various parameters that define the projected trajectory of the sprite. The user is first asked to identify which one of the 16 sprites (numbered 0-15) is to be edited. (Parameter values for all 16 sprites are reset to 0 if the user enters the command "ERASE.") Following the user's response, a table of parameters for the active sprite, referred to as a *Sprite Description*, appears on the screen (see Table 2 for a sample listing of this table). The first parameter is *Status*, which refers to whether or not the sprite will be calculated or displayed. This parameter is controlled by toggling between ON and OFF. The next two parameters specify the *Center of Rotation* for the sprite along the x- and y-axes of the screen. These pa-

Table 2
Sample Listing of a Sprite Description Table

Active Sprite Descriptions	
SPRITE #: 0	STATUS: ON
CNTR OF ROTATION X :	T/180 * 255
CNTR OF ROTATION Y :	95
RADIUS OF CIRCLE :	30
PHASE SHIFT :	90
TILT :	60
ORIENTATION :	45
OSCILLATE :	NO
RANGE OF OSCIL. :	0
ANGULAR DISPLACENT:	6
COLOR :	15
DEPTH PLANE :	0
PATTERN :	PT.LT.

rameters are assigned numerical values corresponding to screen pixels ranging between 0 and 255 on the x-axis and 0 and 191 on the y-axis. The *Radius* of the circle is also specified in screen units. The next three parameters—*Phase Shift*, *Tilt*, *Orientation*—modify the trajectory of the sprite in the ways described previously under the section Underlying Logic of the Design Program. Each parameter is specified by a number corresponding to degrees of arc relative to 0°. Note, that *Tilt* and *Phase Shift* are programmed to change in a counterclockwise direction, whereas *Orientation* is programmed to change in a clockwise direction. *Oscillation* specifies whether or not the sprite oscillates back and forth in an arc, rather than undergoing a complete rotation. This parameter is controlled by toggling between YES and NO. If *Oscillation* is chosen, then *Range of Oscillation* specifies the number of degrees of arc traveled by a sprite in both directions relative to a fixed starting point. (Note—it is programmed to ensure that the cycle time for a complete oscillation is equivalent to the cycle time for a complete rotation.) *Angular Displacement* corresponds to the number of degrees of rotation from one frame to the next. *Color* is specified by a value between 0 and 15. The color white corresponds to the value 15; black corresponds to 1; and transparent corresponds to 0. Other numbers will produce red, green, yellow, and blue, as well as various shades of these colors. The specific correspondence between colors and numbers depends upon certain features of the selected monitor. The *Depth Plane* is specified by a value between 0 and 15, which defines its priority of visibility. Highest priority corresponds to the value 0, and lowest corresponds to the value 15. Finally, the *Pattern* parameter allows the user to choose between a sprite shaped as a point-light or shaped as a rectangle. This parameter is controlled by toggling "P" for point-light or "R" for rectangle. The rectangular shape is typically used as an invisible occluder by assigning it a high priority of visibility and setting its color to that of the background, that is, black.

In addition to specifying a constant for each of the above parameters, it is also possible to enter an algebraic ex-

pression for the first five parameters (center of rotation for x and y, radius, phase shift, and tilt) so that these values are continuously changing from one frame to the next. Each expression may include the following components: positive or negative real numbers, mathematical operators including, +, -, * (multiply), / (divide), and parentheses, which may be used to alter the sequence in which the expression is evaluated. In addition, the following variables may be used: the current frame number (T = frame number), the screen coordinates of a lower numbered sprite (X#, Y#, where # corresponds to a sprite numbered between 0 and 15), that sprite's current angle of rotation, also referred to as theta (TH#), and its phase shift (PS#). Finally, a number of specific functions and logical operators can be incorporated into these expressions. Table 3 contains a listing of these commands.

In order to facilitate rapid and efficient editing of the sprite descriptions, a number of simple editing commands are made available to the user. These commands enable the user to move the cursor up and down the page, as well as to insert and delete characters. The specific characters corresponding to each command appear at the bottom of each sprite description page. (These editing commands are also available during execution of other program options.)

Calculate Animation List

This option requests information from the user concerning the number of discrete frames to be shown and then calculates and stores in RAM the sprite positions for each frame. (Although it would be preferable to display the point-lights as they are calculated, the Apple's microprocessor is not powerful enough to perform the necessary calculations fast enough for producing a real-time display.) The user is requested to enter this information in the following form: first frame to be displayed, last frame to be displayed, and step size, for example, every frame, every second frame, etc. The default values for this information are 0, 179, 2; these values instruct the program to calculate every second frame of the first 180. If the animation sequence is to appear continuous, it is essential that a sufficient number of frames are calculated to ensure displaying a complete cycle of rotation. It is inefficient, however, to calculate more than one complete cycle. The maximum number of frames needed to meet this criterion is 360, but this number can be divided by a factor of 2 or more, depending upon the angular displacement specified in the sprite descriptions. For example, a sprite that is programmed to rotate 4° from one frame to the next would require only 90 frames to produce a complete revolution. Also note, that the step size will affect the number of frames that are calculated, but the user must recognize that setting the step size to a particular value is not equivalent to setting the angular displacement to that same value. Programming a complete cycle is a function of the angular displacement multiplied by the highest frame number calculated; it is not directly affected by the step size. For example, specifying the calculation values 0, 179, 1 will produce a complete cycle

Table 3
Listing of Numeric Functions and Logical Operators

Item	Action
Numeric Function	
SIN ()	Returns the trigonometric sine of the argument (in degrees)
COS ()	Returns the trigonometric cosine of the argument (in degrees)
TAN ()	Returns the trigonometric tangent of the argument (in degrees)
ATN ()	Returns the trigonometric arctangent (in degrees) of the argument
SQR ()	Computes the square root of the argument
ABS ()	Returns the absolute value of an argument
SGN ()	Returns the sign of an argument
INT ()	Returns the integer portion of a floating-point argument
RND ()	Generates a pseudorandom number
LOG ()	Returns the natural logarithm of the argument
EXP ()	Raises the constant "e" to the power of the argument
Logical Operators	
<	Less Than
>	Greater Than
=	Equal To
< >	Not Equal To
> =	Greater Than or Equal To
< =	Less Than or Equal To
AND	Logical And
OR	Logical Or
XOR	Logical Exclusive Or

Note—These operators all return a value of 1 if evaluated as true and a value of 0 if evaluated as false.

if the angular displacement is 2, but changing the step size to 2 and the angular displacement to 1 will produce only half a cycle, even though the same number of frames will be calculated.

Run Animation List

This option enables the user to display the previously calculated frames at a user-specified display rate. Two options are available. Either the user can specify that the display rate will be variable and controlled by a game paddle, or the user can specify the display time for each frame in $\frac{1}{60}$ ths of a second. If the latter option is chosen, the computer will calculate and display the frames-per-second animation rate.

Access Disk

The Disk Access option provides the user with a number of useful disk commands in a menu format: (1) *Catalog* allows the user to catalog the disk in either drive. (2) *Save an Animation List* will create a binary file of the sprite positions for each previously calculated frame and save it to disk. A text file corresponding to the sprite descriptions used for calculating the animation list will also be saved to disk. Procedurally, the user is prompted to enter a base name for these two files, and then the files are automatically created and saved. Although the two files share the same base name, they are distinguishable because the program appends an ".SD" extension to the sprite descriptions file and an ".AL" extension to the animation list file. (3) *Load an Animation List* will load into RAM the binary file specifying the sprite positions for

each previously calculated frame as well as the text file containing the parameters entered and stored for each sprite description. Both files are loaded automatically after the user enters the base name of the two files. (4) *Delete an Animation List* will automatically delete from the disk the two files corresponding to the user specified base name.

Print Sprite Descriptions

This option prints out a list of sprite descriptions for each of the active sprites in memory. The format for each sprite description is identical to the one that appears on the screen in the Enter/Edit option.

Specify Configuration

This option enables the user to modify (1) the correction factor for screen bias and (2) the slot number containing the TMS 9918A VDP card. Both options are available from a menu and are included specifically to provide the user with optimum flexibility in using the Design Program.

Exit

This option returns the user to Applesoft BASIC. It is recommended that the user then execute the DOS command FP to reset all pointers back to their original values.

ILLUSTRATIVE EXAMPLES

The purpose of this last section is to illustrate how specific displays are produced with the Design Program.

Table 4
Distinctive Sprite Parameters for Wheel-Generated Motions

Sprites	0	1	2	3
CNTR OF ROTATION X:	128	XO	XO	XO
CNTR OF ROTATION Y:	95	YO	YO	YO
RADIUS OF CIRCLE :	0	20	20	20
PHASE SHIFT :	0	0	120	240
TILT :	0	60	60	60
ORIENTATION :	0	90	90	90
COLOR :	1	15	15	15

In so doing, we will emphasize how some special effects are produced through the implementation of different algorithms created for the sprite descriptions.

Wheel-Generated Motions

The first display consists of three point-lights moving as though mounted on the rim of an invisible rotating wheel, tilted 60° from the image plane and oriented such that the major axis is rotated 90° from the vertical. Table 4 presents a partial listing of the sprite descriptions used for producing this display. A number of points should be noted about these descriptions. First, it is necessary to specify an invisible center of rotation for all three point-lights. Sprite 0 corresponds to this center and is programmed by setting its radius to 0 amplitude and its color to match that of the background. Second, all three visible point-lights, Sprites 1-3, share the same center of rotation corresponding to the location of Sprite 0. Finally, tilt and orientation are set to the same values for all three visible point-lights, since each light is programmed to follow the same trajectory in depth.

Translation

The second display is designed to simply add a translatory component to the rotating system of lights described in the preceding display. Specifically, the rotating lights are programmed to horizontally translate at a linear velocity across the entire screen from left to right. This effect is achieved by substituting an expression, $T/180 * 255$, for the x coordinate of the center point, Sprite 0. In order to understand how this expression operates, it is necessary to recall that "T" corresponds to the frame number and 180 represents the number of frames to be displayed in this particular example. The product of this proportion $(T/180)$ multiplied by the highest screen coordinate value on the horizontal axis (255) yields a linearly increasing value ranging from 0 to 255 as the frame number increases. As such, this function produces horizontal translation across the screen.

Discontinuous Motions

In the preceding display, the rotating lights were programmed to translate from left to right across the screen and then to wrap around and reappear at the left of the screen. It is also possible to program the lights so that they do not disappear from the screen. The simplest example of such an event would be to program the display to horizontally translate at a linear rate across the

screen and then to reverse direction and return to its starting position. This effect is accomplished by assigning the following expression to the center of rotation for the x coordinate of Sprite 0: $[T/180] * 255 * (T < 90)] + [255 - (T/180) * 255 * (T \geq 90)]$. This expression is actually much simpler than it appears, since the right half reduces to zero if the frame number is 89 or less, and the left half reduces to zero if the frame number falls between 90 and 179. The reason for this result is that a logical expression yields the value 1 if evaluated as true and yields the value 0 if evaluated as false. Accordingly, for Frames 0 through 89, the left half of the expression is multiplied by 1 and the right half is multiplied by 0, whereas, for Frames 90 through 179, the opposite is true. As such, this expression reduces to that previously described for producing horizontal translation from left to right across the screen when only the left half is evaluated. The right half of the expression produces horizontal translation from right to left by progressively subtracting higher numbers from the x-coordinate value corresponding to the right edge of the screen.

Sinusoidal Velocities

Thus far, all the examples involving translation are designed to depict displays moving at a linear velocity. It is sometimes useful to depict these displays as moving at a sinusoidal velocity, especially if the translatory path is not normal to the observer's line of sight. In the case of our current display involving wheel-generated motion, this effect is produced by rotating the center of motion (Sprite 0) for the three visible lights around a very large circle. If this circle is tilted at 90° and is also oriented at 90° , its projected trajectory will be reduced to a horizontal line on the screen with a length equivalent to twice the radius of the circle. Since point-lights are programmed

Figure 2. Sprites 1 and 3 are shown as oscillating along the same projected trajectory around Sprite 0. Sprite 2 is located in a fixed position along the trajectories of Sprites 1 and 3, but is not visible to the observer because it shares the same color as the background. Twice per cycle Sprite 3 will be seen as passing behind Sprite 2, since it possesses a lower priority of visibility than does Sprite 2. Sprite 1, however, will always remain visible, since it possesses a higher priority of visibility than does Sprite 2.

Figure 3. Four points are depicted revolving in a plane slanted in depth. Attached to each of these points is a second point that is swinging with a pendular motion. The tilts of the axes for these pendular motions are continuously changing in direction as a function of the angular rotation of the point to which it is attached.

to move around a circle at a sinusoidal velocity, Sprite 0 will now appear to move back and forth across the screen at a velocity corresponding to a sinusoidal function.

Occlusion

The next display is designed to produce a pair of moving point-lights oscillating in different planes of depth. This effect is achieved by programming one of the point-lights to disappear and then reappear as it passes behind an implicit form located between the two oscillating lights. Figure 2 depicts a static frame from this display. Occlusion is easily produced, since sprites with lower priorities of visibility (higher depth plane values) are automatically occluded by sprites with higher priorities of visibility when portions of the two patterns occupy the same screen positions. For the purpose of the present display, Sprite 2 (with a corresponding depth plane value) was designed as an occluder by setting its color to that of the background (black) and assigning it a rectangular shape. Since the near oscillating point-light, Sprite 1, is assigned a lower depth plane value than the occluding sprite, it does not undergo occlusion, whereas the far point-light, Sprite 3 (with a correspondingly higher depth plane value), does undergo occlusion.

Tilted Pendular Motions

Many of the displays used for studying the perception of jointed objects involve pendular motions in which one point-light oscillates relative to another light. An example of this event is seen in Figure 3, which includes four point-lights rotating around the rim of an invisible wheel at a 60° slant. Each of these point-lights can be described

as the pivot of a pendulum completed by a second point-light oscillating below it. As can be readily observed, the tilt of the pendulum changes as the pivot moves around the rim of the wheel. In fact, the tilt changes as a function of the angular rotation of the pivot in a manner consistent with the physical laws of motion. This effect proves easy to implement with the Design Program because it involves simply setting the tilt for the point-light corresponding to the bob equal to the changing value of θ for the point-light corresponding to the pivot.

It is important to point out that the displays described in this last section were by necessity kept simple. Other displays that we have created using this program are far more complex and depict nested pendular objects rotating and oscillating in depth, as well as large rotating surfaces that comprise as many as 16 point-lights.

We anticipate that this program has many applications beyond our own research. For example, almost all of the demonstrations created by Johansson (1950, 1975) can be recreated using the Design Program. Also, it has proved an excellent tool for producing point-light displays depicting a wide array of rigid, jointed, and nonrigid objects. Finally, the program assures the user not only a great deal of flexibility in the design process, but also provides a powerful analytical tool for assessing the critical parameters in human motion information processing.

REFERENCES

- BARNHILL, R. E., & RIESENFELD, R. F. (Eds.). (1974). *Computer aided geometric design*. New York: Academic Press.
- BERTENTHAL, B. I., & KRAMER, S. J. (1984). The TMS 9918A CDP: A new device for generating displays on a microcomputer. *Behavior Research Methods, Instruments, & Computers*, **16**, 388-394.
- BERTENTHAL, B. I., PROFFITT, D. R., & SPETNER, N. B. (1984). Infants' encoding of kinetic displays varying in relative coherence. *Infant Behavior and Development*, **7**, 34.
- BRAID, I. C. (1978, March). *New directions in geometric modelling*. Paper presented at the meeting of the Geometric Modeling Project, CAM-1, St. Louis, MO.
- CUTTING, J. E. (1978). A program to generate synthetic walkers as dynamic point-light displays. *Behavior Research Methods & Instrumentation*, **10**, 91-94.
- HERMAN, G. T., & WEBSTER, D. (1980). Surfaces of organs in discrete three-dimensional space. In G. T. Herman & F. Natterer (Eds.), *Mathematical aspects of computerized tomography*. Berlin: Springer.
- JOHANSSON, G. (1950). *Configuration in event perception*. Uppsala, Sweden: Almqvist and Wiksell.
- JOHANSSON, G. (1973). Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, **14**, 201-211.
- JOHANSSON, G. (1975). Visual motion perception. *Scientific American*, **232**, 76-89.
- PROFFITT, D. R., BERTENTHAL, B. I., & ROBERTS, R. J. (1984). The role of occlusion in reducing multistability in moving point-light display. *Perception & Psychophysics*, **36**, 315-323.
- PROFFITT, D. R., & CUTTING, J. E. (1979). Perceiving the centroid of configurations on a rolling wheel. *Perception & Psychophysics*, **25**, 389-398.
- REQUICHA, A. A. G. (1980). Representations for rigid solids: Theory, methods, and systems. *Computing Surveys*, **12**, 437-464.
- RESTLE, F. (1979). Coding theory of the perception of motion configurations. *Psychological Review*, **86**, 1-24.