Transactions of the Illinois State Academy of Science
(1990), Volume 83, 1 and 2, pp. 32-40

PC Animation and Scottish Country Dancing

Jayne Schiek and Patrick Lamont
Department of Computer Science
Western Illinois University
Macomb, Illinois 61455

ABSTRACT

If a picture is worth a thousand words, then we may infer that a picture that moves is worth many thousands of words. This premise is the basis for the graphics project presented here. The paper describes the creation of software written in an advanced BASIC to run on an IBM PC to simulate the formations of 32 bars of a Scottish Country Dance. The software serves as an instructional aid helpful to the beginning dancer. Determining the type of representation to be used on the screen, analyzing the various figures to be simulated, and studying techniques of computer animation were among the many interesting facets of solving the problem.

INTRODUCTION

The problem under consideration is that of producing software for Scottish Country Dance (SCD) instruction. Computer graphics animation is used to produce the patterns executed in SCD. Thus the objective of the project is to demonstrate formations of SCD on the computer screen. These formations are described in the definitive work Milligan (1982). The software is to be written to use available hardware, namely, an IBM PC running an advanced BASIC.

The major problem that was encountered in the graphical representation of SCD was the need for extensive calculation for the circular paths which the figures often take. This was solved for the IBM PC representation by setting up "path arrays" which contain the necessary coordinates for the movement of the figures. Using pointers to the array, as well as offsets, eliminates lengthy calculation time during figure movement. The circle and other formation generation techniques employed were chosen to give students practice in mathematics, notably trigonometry.

The formations advance and retire, lead down the middle and up, circle round - right and left, figure of eight, and cast off one are shown in figure 1. These formations are selected from over thirty which are described in Dr. Milligan's work.

The human eye has a characteristic, called persistence of vision, that underlies the success of all animation devices: the retina of the eye retains the image of an object for a brief instant after the removal of the object. The eye blurs many images into one image if the images are presented in quick succession. The computer has recently joined the many devices that have been used to create the illusion of animated pictures. The techniques of computer animation are interesting.

In computer graphics operations the screen is repainted, usually 25 times per second, from a special area of memory, which may be part of the main RAM or may be separate as in the IBM PC. This area is often called the video map because there is a correspondence between the bytes of the video memory and the pixels on the screen. Whatever is put in the video map by the processor automatically appears on the screen the next time it is repainted. Because the screen is repainted so frequently, animation effects can be produced on it.

For the programmer of animation, two systems are available. The first is paging in which there are two or more video maps. This system requires an enormous amount of RAM. The second involves sprites, a number of hardware areas which accept smaller pictures to be transferred into the display area at any given point in time.

The method for implementation of animation is straightforward. The rules are few. An important consideration is the avoidance of flicker. To avoid flicker, the time that a form appears on the screen should be greater than the time the form is off the screen.

IMPLEMENTATION

Keeping these rules in mind, let us turn to the implementation of animation on the IBM PC using BASICA. While there are no sprites available on the IBM PC, the use of GET and PUT statements enable the use of sprite-like areas of RAM. The GET and PUT statements allow the saving of an image or images. The images can be placed on the screen at changing locations. The GET statement transfers the contents of a rectangular area of the screen into an array. The PUT statement is the companion statement to the GET statement, but is not strictly the reverse. An optional parameter, called action, selects one of five ways in which the image may be placed on the screen. PSET and XOR are the most common of the five ways used in producing animation sequences.

The result of PUT with the PSET option is to copy an image drawn with any combination of the PSET, LINE, CIRCLE, PAINT or DRAW instructions to a new screen location without having to redraw the image. PUT with PSET may be thought of as the reverse of GET. This option is quite useful for certain types of animation.

The exclusive OR, XOR, causes the pixels in the image and the corresponding pixels on the screen to be logically XORed together. This allows us to PUT images against a multicolored background. As the PUT image is

moved, the original background is restored. A distracting flicker may occur when erasing and redrawing figures to be animated. One PUT erases the old image and the next PUT draws the new image. During the interval between the two PUT statements, there is no image on the screen causing flicker. It is therefore important to place the two put statements in the program without any intervening operations. When the PSET option is used, the image is PSET initially and then erased by PSETing a blank image on top of it.

If we are not concerned with what is in the background, the PSET option can be used in an even better way. In order to do this, GET the desired image with a border of the background color wider than the distance the image will move each time it is redrawn. Each time the image is redrawn, the border will erase the previously drawn image automatically. Now there is no separate operation to erase the image. Since the image is always on the screen, the animation produced is relatively smooth and flicker free. Action can be faster because there is no need for a separate statement to erase the previously drawn image. However this technique can only be used with a nonpatterned background.

Programming the formations that use motion in straight lines, for example, advance and retire and lead down the middle and up, is quite straightforward using the techniques that have been described. Programming movement in a circular locus presented a more difficult problem.

An early attempt to program a SCD formation that involved movement in a circle included the following program segment.

```
170     FOR I = 1 TO 360 STEP N
180     FOR J = 1 TO 4
190     AN(J) = ( I + 45 + (J - 1)) * P/180
200      X(J) = 96 + R * COS(AN(J))
210     Y(J) = 100 + R * SIN(AN(J))
          .
          .
          .
300     NEXT J
310     NEXT I
```

A large amount of flicker is introduced. The reason for this is the large amount of computational overhead involved in the lines shown. The loop first draws the figures, then the next I iteration erases the figures. This violates the rule that figures should be on the screen for a longer length of time than they are off.

Berger (1984) shows that when a path is known in advance a so-called path look-up table may be used. As paths are known in SCD formations, this method may be utilized. The path is precomputed and the coordinate values stored in a table. Then, whenever the object moves, the look-up table may be accessed to determine the screen coordinates of the object. The principle is that it is quicker to look up a value than to calculate it. Hearn and Baker (1986) present an excellent program for generating points for a circular path and this was adapted for use in a development program. Here is the portion of the program that generates the points around a circle.

```
280     I = 0
```

```
290     XFIRST = XCENTER + RADIUS:
        YFIRST = YCENTER
300     X(I) = XFIRST:   Y(I) = YFIRST
310     I = I + 1
320     FOR ANGLE = DA TO 6.28318 STEP DA
330     X(I) = INT(XCENTER + RADIUS * COS(ANGLE) + .5)
340     Y(I) = INT(YCENTER + RADIUS * SIN(ANGLE) * .8333333 +.5)
350     I = I+1
360     NEXT ANGLE
```

In the preceding lines, the user is prompted to enter the coordinates for the center of the circle, the radius, and the number of points to be plotted around the circular path. The value of 2*pi is divided by the total number of points to be generated to give the size of the angle (DA) in radians which will be used in converting the polar coordinates of the points around the circle into their respective cartesian coordinates. Line 290 fixes the position of the first coordinate. Then the loop in lines 320 through 360, the succeeding points are plotted at DA intervals and stored in the two arrays X and Y. With center (100,100), radius 30, and 60 as the number of points, the look-up table is shown in Table 1.

A program involving circular motion, using the path look-up table follows.

```
100     'PROGRAM: movcirc2
110     ' moves a box around in a circle smoothly
120     XCENTER = 196: YCENTER = 58
130     RADIUS = 32
140     TOTALPOINTS = 48
  .
  .     (calculate path matrix)
  .
260     'initialize screen
270     SCREEN 1,0:COLOR 0,1:KEY OFF: CLS
280     'draw box
290     LINE (10,10)-(17,17),2,BF
300     'save image plus 4 pixels of border
310     DIM BOX1(34): GET (6,6)-(21,21),BOX1
320     CLS
330     CIRCLE (XCENTER, YCENTER), RADIUS,3,,, .8333333
340     ' move square around path
350     'subtraction from corner puts figure on center
360     LOCATE 24,5:  PRINT "Press space bar to move box";
370     FOR I = TO TOTAL POINTS
380     PUT (X(I,-8,Y(I)-8),BOX1,PSET
390     FOR W - : TO 30: NEXT W
400     GOSUB 440
410     NEXT I
420     A$-INKEY$: IF A$ - "" THEN 420
430     SCREEN 0: WIDTH 80: CLS : END
```

```
440     A$=INKEY$:IF A$="" THEN 440
450     RETURN
```

The lines 150 through 250 are the same as in the path array generator program. Now that the path look-up table has been stored, the screen is set to medium resolution graphics mode in line 270. A box is drawn and the image plus 4 pixels of border are saved in lines 280 - 310. A circle representing the path to be traveled is drawn at line 330. Then the user may move the object about this circle using another tool, the subroutine at line 440. This subroutine was found to be most useful in working out the simulation of the dance formations. This "step function" is called at the end of each loop thereby enabling the user to move the object one "frame" at a time. Using this tool, any discrepancies in movement may be observed.

The techniques of using a path array generator for path look-up tables and the step function for examining movement "frame" by "frame" are incorporated into the final project program.

In all of the formations, one iteration of a loop is used to simulate one frame of movement. Following each movement, a wait statement is used which is nothing more than an empty for-next loop, spinning away some time, keeping the image on the screen long enough to impact the eye before moving. The final line of the loop checks to see if the step option has been requested.

Because of space constraints, just one of the circular formations, the Circle Round Right and Left, will be discussed. This formation requires the apparent movement of four figures at one time.

The position of the path was determined using graph paper. A radius of 20 pixels was determined to be a good size for the figure and using an approximately 2 to 3 ratio between radius and number of points to be plotted, 2 was chosen as the number. Note that in all cases a number which is divisible by four has been chosen in order to make the circle easy to segment.

Once again, the path array was printed out, so that the positions in which each figure would start to circle could be ascertained. Using these positions as a goal, the path for each image to travel from the start position was determined. How many steps to take? Just draw the circle on the screen, use the step function, until the position is reached and count!

When each of the formations had been developed, we were ready to assemble the program. Each module was joined to the main shell of the program using the advanced BASIC append file function. The hierarchy chart in figure 2 shows the program organization.

CONCLUSION

The project introduced us to the animation capabilities of the IBM PC. One of the major questions which was raised at the start of this project was whether or not the use of advanced BASIC on the IBM PC would be a suitable tool for implementing the formations as an instructional tool for the beginner to learn Scottish Country Dancing. The completed project shows evidence that this implementation is a workable solution to the problem.

LITERATURE CITED

Berger, Marc, Computer Graphics with Pascal, Menlo Park, California (1984).

Hearn, Donald and Baker, M. Pauline, Computer Graphics, Englewood Cliffs, New Jersey (1986).

Illowsky, Dan and Abrash, Michael, Graphics for the IBM PC, Indianapolis (1984).

Markowsky, George, A Comprehensive Guide to the IBM Personal Computer, Englewood Cliffs, New Jersey (1984).

Milligan, Jean C., Won't You Join The Dance? The Scottish Country Dance Manual, Edinburgh and London (1982).

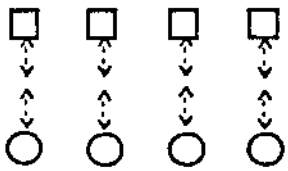
Table 1. A path look-up table.

Center Point: 100 100
Radius: 30
Number Points: 60

	0		1		2	
	x	y	x	y	x	y
0	130	100	130	103	129	106
6	124	116	122	118	120	121
12	109	126	106	127	103	127
18	91	126	88	125	85	124
24	76	116	74	114	73	111
30	70	100	70	97	71	94
36	76	84	78	82	80	79
42	91	74	94	73	97	73
48	109	74	112	75	115	76
54	124	84	126	86	127	89
60	130	100	0	0	0	0

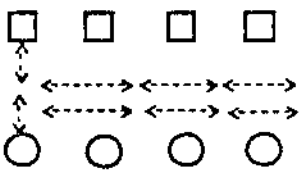
	3		4		5	
	x	y	x	y	x	y
0	129	109	127	111	126	114
6	118	122	115	124	112	125
12	100	128	97	127	94	127
18	82	122	80	121	78	118
24	71	109	71	106	70	103
30	71	91	73	89	74	86
36	82	78	85	76	88	75
42	100	72	103	73	106	73
48	118	78	120	79	122	82
54	129	91	129	94	130	97
60	0	0	0	0	0	0

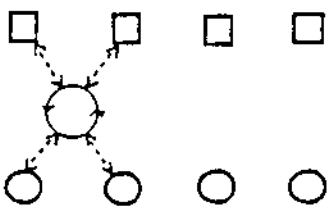
Figure 1. Scottish Country Dance Formations.



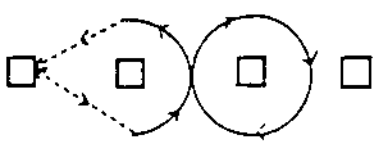
1. Advance and Retire

2. Down the Middle and Up

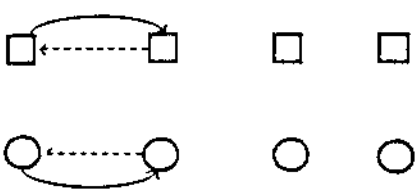




3. Circle Round



4. Figure of Eight



5. Cast off One

Figure 2. Hierarchy Chart: Scottish Country Dance Formations.