A COMPUTER PROGRAM TO DISPLAY ANIMATIONS WITHIN THE COMPUTER GRAPHICS HALO ENVIRONMENT

Efraim Halfon and David De Jong

NWRI Contribution No. 89-20

Lakes Research Branch
National Water Research Institute
Canada Centre for Inland Waters
Burlington, Ontario
Canada L7R 4A6

May 1989

MANAGEMENT PERSPECTIVE

This report describes a numerically efficient program to display computer graphics animations on IBM compatible microcomputers. This report is distributed with a floppy disk containing the appropriate software with a demonstration of the movement of a toxic contaminant plume in Lake St. Clair.

PERSPECTIVE-GESTION

Le présent rapport décrit un programme de numérisation efficace qui affiche des images graphiques animées sur un microordinateur compatible IBM. Le rapport est fourni avec une disquette renfermant le logiciel approprié et une démonstration du mouvement d'un panache de contaminant toxique dans le lac St. Clair.

ABSTRACT

This subroutine is designed to be used in any case where the user is working with a digitized image and is modifying the image, but not according to any kind of recognizable pattern. The subroutine was originally designed to be used with an image block which represents the concentrations of chemicals throughout a large body of water.

In this paper we explain how HALO stores a graphic screen and therefore how any screen can be created from any matrix. One additional option is that a zoom factor is included so that any part of the matrix can be displayed enlarged.

RÉSUMÉ

Ce sous-programme peut être utilisé lorsque l'utilisateur travaille avec une image numérisée et qu'il veut modifier l'image selon un schéma non reconnaissable. Ce sous-programme devait à l'origine être utilisé avec une image représentant les concentrations de produits chimiques dans une grande étendue d'eau.

Dans cet article, nous expliquons comment le programme HALO stocke une image graphique et comment on peut créer une image à partir de n'importe laquelle matrice. On peut choisir un zoom pour grossir n'importe laquelle partie de la matrice.

INTRODUCTION

Animation is a useful tool to display the dynamical behaviour of computer simulations. Animation is relatively easy to program on microcomputers given appropriate software. One software program quite suited to this purpose is HALO (1986). Nevertheless, even in its sophistication, HALO lacks a subroutine, crucial for the successful display of computer simulations through animation. This paper presents an analysis of the procedure HALO employs to store screen images in memory and it shows the development and application of the missing routine which we call CONVRT. This routine transforms the numbers produced by a mathematical model to colour display on the graphics screen within the HALO environment.

THE HALO LIBRARY

The HALO library is a collection of high performance subroutines which allow a programmer to implement sophisticated computer generated graphics. The library consists of over 170 graphic functions written in assembler and supplied to the user in object code form, ready to be linked with a high level language. The HALO library is a very useful tool since the IBM personal computers, models XT and AT, are not very well adapted to display animated computer graphics. Furthermore, animation has two important requirements, speed of computation and image resolution. The greater the resolution required, the higher the amount of computation needed to draw an image.

HALO can be used with a variety of programming languages, FORTRAN, BASIC, C, etc. Although the routines offered are fairly fast (being written mostly in assembly language), if the generation of an image requires many

calculations, the computer animations might execute slowly. A solution is to calculate the entire image first, store it in memory and then display it on the graphics screen all at once.

GRAPHICS BOARDS

IBM microcomputers rely on graphic boards to create a graphic screen. Every graphic board displays only a finite set of colours. This finite set of colours is referred to as a "palette." Some boards contain multiple palettes and have predefined sets of colours that cannot be changed. The IBM Color Graphic Adapter, CGA for short, has two palettes, each with a preset group of four colors. More advanced boards, such as the EGA and VGA allow the display of 16 out of 64 colours and 256 colours, respectively.

In this paper we describe a subroutine that is applicable to CGA boards but easily expandable to EGA and VGA boards, once the principles are understood. The IBM CGA has two palettes, palette 0 has the four colors, black (with index or number value 0), green (with index or number value 1), red (with index 2) and brown (with index 3). Palette 1, which will be used in the following examples, has black (index 0), cyanide or light blue (index 1), magenta or purple (index 2) and white (index 3).

On a CGA screen each pixel can have one of four colors. In the decimal system the indices vary between 0 and 3. In the binary system, used by computers, the four colors can be represented by any one of two bits combinations, for example zero [0,0] is black, one [0,1] is light blue, two [1,0] is magenta and three [1,1] is white.

PRINCIPLES OF COMPUTER ANIMATION

Animation requires a fast refreshment of the computer screen, to show the changes that take place in time. For example the display of the movement of a contaminant plume in a lake implies several steps:

a) Numerical solution of the mathematical model.

b) Storage of the simulation results in a matrix, MATR. Each element of MATR contains the concentrations of a contaminant at a grid point, spaced, for example, two kilometres apart.

c) Classification of the matrix values into ranges. This step implies the conversion of the raw simulation data in matrix MATR to data in classified form in matrix INP according to the indices of colours. The matrix INP is a two-dimensional array which represents an image by storing the colour of each pixel according to an X-Y coordinate system in which X-positive is to the right and Y-positive is down. Since only a fixed number of colours are allowable on the screen (in case of a CGA board only four colors can displayed at the same time), each colour must represent a range of concentrations. For example (Table 1) if a simulation shows that a contaminant has concentrations of less than 5 ng/L in water, then the screen colour is black (identified by the number 0). If concentrations are between 5 and 10 ng/L the colour is white (identified by the number 3); if concentrations are between 10 and 50 ng/L the colour is light blue (identified by the number 1), and if concentrations are higher than 50 ng/L the colour is magenta (identified by the number 2). Each element of the array INP now contains values between zero and three.

d) The contents of the matrix INP are transformed into a vector IMG that can be understood by HALO subroutines, specifically subroutine MOVETO, to display the colour image to the screen.

e) Call to subroutine MOVETO to display colour pixels on the screen.

f) Steps a-e must be repeated for each time step so that the animation can take place. At each time step a new matrix INP is created. In this example the contaminant plume moves across the screen.

It is clear that many computations must take place at each time step. This paper focuses on the step (d). The HALO manual provides no clues on the structure of the array IMG and the user assistance office provided no help. Since other HALO users might be interested in showing computer animations, procedure (d) is explained here.

THE MISSING SUBROUTINE: CONVRT

HALO provides a number of commands for moving portions of the screen as well as storing these images in memory or in an array. The storing of these images is done efficiently by using the minimum number of bits necessary to represent the number of colours available at any one time (i.e. four colours are represented by two bits). Even if HALO contains many graphics subroutines useful to the graphics programmer, the subroutine CONVRT identified in (d) above is missing. Among its 170 subroutine HALO offers two routines, MOVEFROM and MOVETO, that are closely related to the problem at hand. Routine MOVEFROM copies a portion of the screen to a vector IMG. The assumption is that the graphic screen was created by any of the other 168 routines. HALO developers therefore incorrectly assumed that the contents of the vector IMG are of no

interest to the user. This information is not provided in the HALO manual and was found by extensive investigation of computer images.

Animation requires a subroutine that shows in colour on the computer screen a digitized picture stored in computer memory as a matrix, or as a two-dimensional array. This two-dimensional array, INP, stores the colour of each pixel of the digitized image. Routine MOVETO restores a portion of the screen from the array IMG. By an appropriate use of subroutine MOVEFROM and MOVETO, a HALO user can move portions of the screen to a different location or copy parts of the screen. With these routines a user can not modify the contents of the array IMG as required in an animation procedure. To modify the array IMG we must first understand the meaning of each element of IMG.

STRUCTURE OF THE ARRAY IMG IN ROUTINES MOVEFROM, MOVETO AND CONVRT

The simplest representation of a digitized image is a two dimensional array where each array element represents a pixel according to its colour. Since the IBM XT and AT are 16 bits machines, HALO compacts eight pixels into a two-byte vector as is explained in more detail later. Unfortunately, the value of the two-bytes vector, which in 16-bits machines is a number between -65535 and 65535, does not have a one to one correspondence with the two-bytes vector of pixels on the screen. Thus, one of the purposes of this work is to investigate how HALO stores all the pixels of a picture in memory (vector IMG), and therefore how we can use this information to display animation on an IBM PC computer screen.

A careful analysis of computer and screen memory produced the following structure for the vector IMG. The first two bytes of IMG contain the integer

width of the image in pixels or the number of columns of the array INP. The second two bytes in IMG contain the integer height of the image in pixels, or the number of rows in INP. The fifth and sixth bytes are always zero. The seventh and eight bytes contain the first eight pixels of the image to be displayed, left to right, starting at the top of the image. This means that the bytes following the first six contain the pixels of the picture in a packed form using units of two-bytes, or 16 bits.

The next problem is the understanding of the way HALO stores a two byte integer and its relation to the image on the screen. A careful investigation showed a complex pattern that can only be explained with an example: Assume that on the screen we want to show eight horizontally contiguous pixels of the colours black (0,0), blue (0,1), magenta (1,0), white (1,1), white (1,1), magenta(1,0), blue (0,1) and black (0,0). The two-bytes vector on the screen is therefore

high byte low byte

[0 0, 0 1, 1 0, 1 1 ! 1 1, 1 0, 0 1, 0 0]. (1)

To display this series of eight pixels HALO uses a number between -65535 and 65535. In HALO memory the high and low bytes are switched, thus the vector (1) now becomes

former low byte former high byte

[1 1, 1 0, 0 1, 0 0 ! 0 0, 0 1, 1 0, 1 1]. (2)

This 16 bits vector is converted to a decimal number. In this particular example the first digit is a 1, thus the decimal number is negative. In microcomputer arithmetic a negative number is its 2's complement. The complement of (2) is

[0 0, 0 1, 1 0, 1,1 ! 1 1, 1 0, 0 1 ,0 0]. (3)

The numerical value of the integer represented in this byte (3) is

- (0*32768 + 0*16384 + 0*8192 + 4096 + 2048 + 0*1024 + 512 + 256 + 128 + 64 + 32 + 0*16 + 0*8 + 4 + 0*2 + 0*1) = -7140.

This integer is now stored into an element of IMG. Each element of IMG describes a series of eight pixels on the screen from left to right and then down the rows.

One further complication arises if the image is not an integral multiple number of four pixels wide. In this case the image must be padded with blank pixels (zeros) at the end of each row. These extra zeros shall be referred to as the "fill column." A two-byte integer can be split across rows but a single byte may not be split. When the image is finally put on the screen these extra zeros of the "fill column" do not affect the shape of the image since the first element, IMG(1), of the vector specifies the real width of the image.

FEATURES OF THE SUBROUTINE CONVRT

Our subroutine CONVRT computes the numerical value of each element of the vector IMG given a matrix INP, whose contents are explained above. The subroutine also takes into account the position of the fill column so that the display on the screen agrees exactly with the contents of the matrix INP.

One last problem arises for a faithful representation of the matrix of the screen, the problem of scale. A CGA screen in four colours has 320 pixels horizontally and 200 vertically. EGA and VGA screens have even a higher resolution. Each pixel is therefore very small and even a large INP array,

for example 92 columnsby 20 rows would occupy a small portion of the screen, the rest being left black. The solution is to have a zoom factor, so that for example a 92 x 20 matrix can be displayed as a 184 x 40 picture (zoom factor of 2) or as a 276 x 60 picture (zoom factor of 3). In this case a zoom factor of four would be impossible since it would produce a screen picture of 368 x 80 pixels, larger than the CGA screen of 320 x 200.

DISCUSSION

Animation has two important requirements, speed of computation and image resolution. In this paper we focused on improving the speed of computation in building an image to displayed on a CGA screen. Given the assumption that a mathematical model is efficient in producing the matrix INP in real time, or that the matrix INP is read in from a hard disk at each time step, the limiting factor becomes the conversion of the image computed from the simulation to a format that can be shown on the screen. Furthermore, the larger the image, the higher the amount of computation needed to draw an image. A computer user is usually willing to wait for one to three seconds for a picture to be updated on the screen before becoming restless. The subroutine CONVRT is optimized for speed and even with the zooming factor taken into consideration, speed is maintained. Also, programmers designed the HALO routine MOVETO, written in assembly language, for speed. The graphics screen is therefore refreshed regularly and promptly and smooth animation of computer simulations is possible.

The software presented with this paper shows a computer simulation of the flow of toxic contaminants in Lake St. Clair following a hypothetical

spill in the St. Clair River. The size of the array INP is 23 x 23 pixels, zoomed with a factor of 5 to 115 x 115.

REFERENCES

HALO, 1986. Graphics Kernel and Device Drivers. Version 2.26a. Functional Description Manual. Media Cybernetics Inc., 8484 Georgia Avenue, # 200, Silver Spring, MD 20910, U.S.A.

Table 1: Relation among concentrations, contents of the matrix INP, representation of each pixel in the matrix IMG and screen colours.

Contaminant Concentration [ng/L]	INP value (decimal)	pixel value (binary)	screen colour
< 5	0	[0,0]	black
5 - 10	3	[1,1]	white
11 - 50	1	[0,1]	light blue
> 50	2	[1,0]	magenta

```
CC  ********************************************************************************
C
C   SUBROUTINE CONVRT
CC
C   PURPOSE
C      This subroutine creates a vector, IMG, which can be passed directly
CC     to the MOVETO Subroutine provided by HALO (Version 3.26).  The input
C      is the two dimensional array INP (NCOLS X NROWS) which represents an
C      image by storing the colour of each pixel according to an X-Y
CC     coordinate system in which X-positive is to the right and Y-positive
CC     is down.  A zoom factor can also be supplied to scale the image.
C
CC  USAGE
CC     CALL CONVRT(INP,IMG,NCOLS,NROWS,IFAC,IER)
C
C   DESCRIPTION OF PARAMETERS
CC
C       INP -    TWO DIMENSIONAL ARRAY OF NCOLS COLUMNS AND NROWS ROWS
C                THE ARRAY CONTAINS THE COLOUR VALUES (between 0 and 3)
CC      IMG -    THE OUTPUT VECTOR (EIGHT PIXELS PER ARRAY ELEMENT)
C                (DIMENSION CALCULATED - MUST BE SET TO AT LEAST
C                (NPIXTOT/8+3) IN MAIN PROGRAM)
CC               IMG(1) = NCOLS*IFAC
CC               IMG(2) = NROWS*IFAC
C                IMG(3) = 0
C                IMG(4,...,(NPIXTOT/8+3)) = A SIGNED INTEGER REPRESENTING
CC                                          EIGHT PIXELS
C       NCOLS - NUMBER OF COLUMNS IN INP
C       NROWS - NUMBER OF ROWS IN INP
CC      IFAC  - ZOOM FACTOR ( >= 1 )
C       IER   - 0 NORMAL TERMINATION
C               1 IF IFAC IS LESS THAN 1
CC              2 IF NCOLS*IFAC IS GREATER THAN 320 OR NROWS*IFAC IS
CC                GREATER THAN 200 (I.E. GREATER THAN CGA ALLOWS)
C               3 IF A COLOUR IS GREATER THAN 3 OR LESS THAN ZERO
C
CC  REMARKS
C      NO ACTION IF IER EQUAL TO 1 OR 2.
C      IF COLOUR IS OUT OF RANGE COLOUR VALUE IS SET TO ZERO.
CC     NBYTES IS THE IMAGE WIDTH IN SEGMENTS FOUR PIXELS WIDE.
C      NPIXTOT IS THE TOTAL NUMBER OF PIXELS IN THE IMAGE INCLUDING
C      THE FILL COLUMN.
CC     THIS SUBROUTINE IS GENERALLY USED DIRECTLY BEFORE CALLING
CC     SUBROUTINE MOVETO.
C
CC  SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
CC     NONE
C
C   METHOD
CC     THE VECTOR INPV IS THE ONE DIMENSIONAL EQUIVALENT OF THE TWO DIMENSION-
C      AL ARRAY INP.  ALL ELEMENTS ARE INITIALIZED TO ZERO, SINCE EACH ROW MAY
C      NEED TO BE PADDED WITH ZEROS IF NCOLS*IFAC IS NOT AN INTEGRAL MULTIPLE
CC     OF FOUR.  THE VECTOR, IMG, IS THEN GENERATED BY COMBINING EIGHT ARRAY
CC     ELEMENTS OF INPV FOR EACH ELEMENT IN IMG USING SIGNED, 2 BYTE BINARY
```

```
CCC     ARITHMETIC.  THE FIRST TWO ELEMENTS OF THE VECTOR IMG CONTAIN
CC      THE DIMENSIONS OF THE IMAGE AND THE THIRD ELEMENT MUST BE ZERO,
CCC     AS REQUIRED BY SUBROUTINE MOVETO.
CCC
CC
CC  ***************************************************************************
CCC
      SUBROUTINE CONVRT(INP,IMG,NCOLS,NROWS,IFAC,IER)

      INTEGER*4 IA,I,ITMP,INDEX
      INTEGER NCOLS,NROWS
      DIMENSION INP(NCOLS,NROWS),IMG(*)
      DIMENSION INPV(65000)
      DATA INPV/65000*0/

      IF(IFAC.LT.1) THEN
        IER = 1
        RETURN
      ENDIF

C     CALCULATION OF DIMENSIONS OF IMAGE
      IMG(1) = (NCOLS*IFAC)
      IMG(2) = (NROWS*IFAC)
      IMG(3) = 0

      IF(IMG(1).GT.320 .OR. IMG(2).GT.200) THEN
        IER = 2
        RETURN
      ENDIF

C     GENERATION OF INPV
C     INDEX GIVES THE ARRAY ELEMENT FOR THE VECTOR INPV WHICH CORRESPONDS
CCC   TO THE ELEMENT (II,JJ) IN THE ARRAY INP.

C     NBYTES IS THE IMAGE WIDTH IN SEGMENTS WHCIH ARE FOUR PIXELS WIDE.
      NBYTES = INT(FLOAT(NCOLS*IFAC)/4.0 + 0.9999)
      DO 11 JJ=0,NROWS-1
       DO 11 II=0,NCOLS-1
        DO 11 K=1,IFAC
         DO 11 L=1,IFAC
          INDEX = (II*IFAC + K) + (JJ*IFAC + L-1)*NBYTES*4
          IF(INP(II+1,JJ+1).GT.3 .OR. INP(II+1,JJ+1.LT.0)
            INP(II+1,JJ+1)=0
            IER = 3
          ENDIF
          INPV(INDEX) = INP(II+1,JJ+1)
   11 CONTINUE

CC    NPIXTOT IS THE TOTAL NUMBER OF PIXELS IN THE IMAGE INCLUDING THE
CC    FILL COLUMN
      NPIXTOT = NBYTES*4*NROWS*IFAC
```

```
CCC   GENERATION OF IMG
CC    IA IS THE REFERENCE ELEMENT FOR EACH BLOCK OF EIGHT VALUES IN INPV.
CC    THE EIGHT VALUES FROM INPV REPRESENT THE 16 BITS WHICH ARE USED TO
CCC   CALCULATE A VALUE FOR IMAGE.  SIGNED INTEGER ARITHMETIC IS USED.

      DO 12 I = 1,NPIXTOT,8
       IA = I+7/8 + 3
CCC   POSITIVE OF NEGATIVE INTEGER?
       IF (INPV(I+4).GE.2) THEN
        ISGN = -1
        ICL  = 3
       ELSE
        ISGN = 1
        ICL  = 0
       ENDIF
CC     EIGHT VALUES FROM INPV COMBINED INTO ONE VALUE
       IMG(IA)= (IABS(INPV(I+4)-ICL)*16384 + IABS(INPV(I+5)-ICL)*4096
     1         + IABS(INPV(I+6)-ICL)*1024  + IABS(INPV(I+7)-ICL)*256
     2         + IABS(INPV( I )-ICL)*64    + IABS(INPV(I+1)-ICL)*16
     3         + IABS(INPV(I+2)-ICL)*4     + IABS(INPV(I+3)-ICL)
     4         + ICL/3 )*ISGN
12    CONTINUE

      RETURN
      END
```