Behavior Research Methods, Instruments, & Computers
1989, 21 (5), 548-563

Programs for visual psychophysics on the Amiga: A tutorial

STUART ANSTIS
York University, North York, Ontario, Canada

and

MICHAEL PARADISO
Smith-Kettlewell Eye Research Foundation, San Francisco, California

A typical psychophysical experiment presents a sequence of visual stimuli to an observer and collects and stores the responses for later analysis. Although computers can speed up this process, paint programs that allow one to prepare visual stimuli without programming cannot read responses from the mouse or keyboard, whereas BASIC and other programming languages that allow one to collect and store observer's responses unfortunately cannot handle prepainted pictures. A new programming language called The Director provides the best of both worlds. Its BASIC-like commands can manipulate prepainted pictures, read responses made with the mouse and keyboard, and save these on disk for later analysis. A dozen sample programs are provided.

Psychophysics is much easier with the aid of graphics on a home computer (Cavanagh & Anstis, 1980), and the Commodore Amiga is one of the best home computers for colored graphics. The Amiga has some advantages over the more expensive Macintosh II, in that it can hold many more pictures in memory to enable page flipping. The Amiga has only 16 gray levels (4 bits per gun per pixel), whereas the Mac II has 256 (8 bits), but any Mac picture must be specified in the full 256-color format whether you want it or not, which uses up 640K of memory, so only one whole picture can be stored at a time, which rules out rapid page flipping. By specifying Amiga pictures with only the number of colors one really needs (2, 4, 8, 16, or 32), one can squeeze up to 50 pictures into 512K of RAM. Pictures can be flipped and moved around rapidly with the Amiga's built-in blitter (block image transfer) chip. Although there are some excellent paint programs for the Mac II, including the remarkable Studio/8 (published by Electronic Arts, 1820 Gateway Drive, San Mateo, CA 94404), there is nothing available like The Director to manipulate them within a psychophysical program. Add-on boards such as the NuVista make the Mac II very powerful but also very expensive. The inexpensive Amiga offers many of the graphics features of the Mac II for a fraction of the price.

Visual stimuli can be painted on the Amiga screen with the commercial program DeLuxePaint II (henceforth called DPaint for short; also available from Electronic Arts). Methods for using DPaint in visual psychophysics have been described in an earlier article (Anstis, 1986). The latest version (DPaint III) also permits page-flipping animations.

A new program greatly increases the usefulness of the Amiga. This is The Director (The Right Answers Group, Box 3699, Torrance, CA 90510, (213)-325-1311). Essentially the program provides a BASIC-like language for manipulating the pictures produced by DPaint, including whole and partial-screen page flipping, turning on and off the DPaint facilities such as transparency and color cycling, collecting responses from the subject in the form of inputs from the mouse and keyboard, and SAVEing these to disk for later analysis. These abilities make DPaint and The Director an ideal combination for running psychophysical experiments. It is also worth buying The Director Toolkit, a disk that is sold separately but cannot be used on its own.

An Amiga 500, 1000, or 2000 is necessary, with two floppy disk drives: the internal drive known as df0:, and an external drive, df1:. Some extra RAM memory is preferable as well. A hard disk (dh:) is convenient but not essential. It is best to buy rather than borrow the program disks, since apart from ethical considerations it is essential to own and study the manuals. One should make working copies of the DPaint and Director disks and write-protect them by moving the tag in the corner of the disks.

This article is not a substitute for the manuals that come with the programs. The reader is assumed to have some familiarity with the Amiga and with DPaint; if not, one should study them beforehand. The programs listed here will help introduce one to The Director, but the manual should always be consulted along with the information presented here. Proficiency will come only from careful study and practice.

This work was supported by Grant A 0260 from the Natural Science and Engineering Research Council of Canada (NSERC). Thanks to Shin Shimojo, to Dorothy Kleffner, and to Maureen Reed for criticizing the manuscript. The authors have no personal or financial interest in any of the commercial products mentioned. Correspondence should be addressed to Stuart Anstis, Department of Psychology, York University, 4700 Keele St., North York, Ontario M3J 1P3, Canada.

This article does two things. For graphics and animation, the main strengths of The Director, it provides simple working programs that will guide the user. These can be modified at will, but for a full understanding of the graphics functions one should work through the manual. In areas where The Director is not so strong—the handling of strings, keyboard inputs, data, and files—this article provides more extensive coverage in order to adapt The Director specifically for use in psychophysical experimentation.

The following programs are included:

showpix	Moves pictures around on the screen.
Muller	A staircase procedure to measure the Müller-Lyer illusion.
fillarray	Accepts keyboard input.
saveinput	Saves keyboard input to disk.
readinput	Reads data from disk.
savenumbers	Saves keyboard input on disk.
(mean&SD	A BASIC program to calculate means & SDs.)
readnames	Reads picture names from disk.
mousepalette	Mouse controls palette colors.
keypointer	Keyboard controls a pointer on the screen.
aspect.ratio	Complete experiment on peripheral acuity.

Getting Started

DPaint II or The Director should always be in the internal disk drive; the program disk should be in the external drive. First, the Workbench disk should go into the internal disk drive (df0:) and a new, unformatted disk into the external drive (df1:). Initialize the disk in df1:. The program disk can be given any name desired. Drag to it a copy of the Empty drawer from the Workbench disk. Rename this drawer from "copy of Empty" to "pix". Drag a copy of the CLI (Command Line Interface) icon from the Workbench to the program disk. The Workbench disk should now be removed and DPaint inserted into df0:. Now some sample pictures are going to be SAVEd on the program disk for use in the Director programs described below. Drag the icons of the pictures called Seascape, Moondance, and Starflight from DPaint into the pix drawer on the program disk. (Note: Different versions of DPaint may provide different pictures on the art disk—earlier versions may have KingTut, Waterfall, and Starflight. If so, use these; it does not matter which demonstration pictures one uses. Simply remember to change the picture names in the programs.) Now open DPaint, setting the resolution to lo-res and the number of colors to 32. DPaint will open to a blank black painting screen with the usual array of painting tools. Do not paint anything yet. Instead, SAVE the blank picture to df1:pix under the name "blank32." Pull down the leftmost menu, select "Screen Format," and change the number of colors in the palette from 32 to 4. This will conserve space when pictures are stored in RAM memory and also when stored on disk.[1]

SAVE this 4-color blank screen under the name "blank4." Note that "blank4" has a 4-color palette, while "blank32" has a 32-color palette and must be used whenever any other 32-color pictures are also loaded. Return to the DPaint painting screen, still with a 4-color palette, and select the color red and the 5×5 square as the brush. Hit F9 (top right of keyboard) to get rid of the top menu bar and click down a red square at the extreme top left corner of the DPaint screen. Put a 1-pixel black dot in the middle of the red square. SAVE this screen to df1:pix under the name "spot." Make sure that it is SAVEd into the pix directory; that is, in the SAVE dialog box, type "df1:pix" in the Drawer box and type "spot" in the File box. Then click on the Save box. Returning to the DPaint screen, some diagonal gratings will now be drawn. Clear the screen to black, select the white color, the 3×3 square brush and the straight-line tool, and set the Grid to 4×4. Draw a 45° diagonal grating of equally spaced black and white lines sloping down to the right (from 10 o'clock to 4 o'clock) covering the entire screen. Hit F10 to hide the menu bars and ensure that the stripes cover these regions too. SAVE this grating to df1:pix under the name "surround." Set the background color to red (to turn off the transparent feature) and pick up the entire grating with the brush selection tool. Reverse it left-to-right by hitting the X key and print it down. This gives a grating sloping down to the left, from 2 o'clock to 8 o'clock.

SAVE this to df1:pix under the name "target." With the transparency still turned off, pick up a random rectangle, say about 1 in. wide and 2 in. high, with the brush selection tool. Hit X to reverse it left-to-right and print it down in a random position. Continue to pick up rectangles of random sizes and shapes, reverse them, and print them down in random positions, until you have an irregular zigzag pattern over the whole screen. SAVE this to df1:pix under the name "mask." Finally, CLEAR the screen again and draw a Müller-Lyer arrow figure, exactly as shown in Figure 1. Turn on the coordinates from the DPaint Preferences menu to ensure that the *x*, *y* coordinates of your drawing are exactly correct. SAVE this 4-color picture to df1:pix as "arrow."

To check that all the pictures are correctly saved, click SAVE again in the leftmost DPaint menu. In the SAVE dialog box, something like this should appear:

```
Parent(dir)
pix(dir)
```

Clicking on pix(dir) should give, in alphabetical order:

```
arrow
blank4
blank32
mask
Moondance
```

Seascape
spot
Starflight
surround
target

Nothing more needs to be SAVEd, so click on Cancel; then, QUIT DPaint. Remove the DPaint disk from df0:, and replace it with The Director disk.

The Director does not use the Intuition user interface with its convenient point-and-click icons. Instead, it uses the CLI (command line interface), which is a part of AmigaDOS. The elements of CLI should be studied in the AmigaDOS manual (1988) or in Levy (1986). For the time being, remember that

```
1>cd df0:
```

sets the current directory to the internal disk drive df0:, and

```
1>cd df1:
```

sets the current directory to the external disk drive df1:.

Graphics Programs

Starting from the familiar blue Workbench screen, the user should double-click on the icon of The Director disk, then double-click on the CLI (or Shell) icon to enter CLI. A blue window with a prompt will appear:

```
1>
```

Programs will be written and edited (modified) using ED, the Amiga screen editor. The rudiments of ED should be studied in one of the AmigaDOS reference manuals. A few of the more useful ED commands are listed in the Appendix. For now, when the prompt appears, type

```
1>cd df1:
1>ed showpix
```

A blue window will appear, with "Creating a new program" in orange text at the bottom. The following program should then be typed:

```
REM showpix: the moving window
REM the next line means "load into buffer 2 the Seascape picture which is in the
REM pix directory on the external disk drive df1:"
REM A buffer is a region of memory allocated to hold a picture.
LOAD 2,"df1:pix/Seascape"
LOAD 1,"df1:pix/blank32"            :REM not "blank4" which has only a 4-color palette
LOAD 3,"df1:pix/Moondance"

FIXPALETTE 2                        :REM stay with Seascape palette
DISPLAY 1                           :REM display buffer 1
BLITDEST 1                          :REM destination for all blits (=picture moves) is
                                     buffer 1
BLIT 2,0,0,0,0,320,200              :REM copy Seascape from buffer 2 to buffer 1
xc=160: yc=100                      :REM center of screen

REM To "blit" means to transfer a rectangular area of picture from one buffer to another
REM Line 10 copies a 70×70 window from Moondance in buffer 3 and moves it across Seascape
FOR dx=0 TO 100
   10 BLIT 3,100,100,dx,50,70,70    :REM buffer,fromx,fromy,tox,toy,width,height
   12 BLIT 2,dx-10,50,dx-10,50,10,70  :REM repair trailing edge of window
   PAUSE 1
NEXT dx
FIXPALETTE 3                        :REM change to Moondance palette
GETKEY dummy                        :REM program waits until one hits any key
```

To exit from ED, the ESCape key (top left on keyboard) should be hit, followed by X, then <CR>. The program will automatically be saved to df1: (the current directory) under the name showpix. (To exit from ED without saving edits, use ESCq<CR>.)

To run your program from CLI, type

```
1>cd df0:                    [DOS will now look on df0: for the Director program]
1> director df1:showpix      [Showpix, like all our programs, is on df1: Do not type:
                              RUN Showpix, RUN is a command in BASIC and
                              does not exist in the Director language.]
```

A square window of size 70×70 pixels containing part of the Moondance picture will slide from left to right across Seascape. Line 12 repairs the trailing left-hand edge of the window, inserting a vertical stationary slice 10 pixels wide from Seascape into the footprint of the Moondance window. Delete line 12 and run the program again (you should know how to do this by now). The window will now smear a trail behind it. FIXPALETTE 2 stays with the palette of buffer 2, namely Seascape. Each picture has its own palette SAVEd with it, but the Amiga can display only one of these palettes at a time, so if two pictures have different palettes, one picture will be inappropriately colored by the other palette. Click the mouse and hit any key to exit.

One can now modify the program without destroying the original. From CLI, type

```
1>copy showpix showpix2
```

This makes an identical copy of showpix and SAVEs it under the name showpix2, leaving the original version of showpix intact. Type

```
1>ed showpix2
```

From within ED, change REM showpix to REM showpix2, and replace line 10 with

```
10 blit 3, xc-dx, yc-dx, xc-dx, yc-dx, 2*dx, 2*dx
```

Exit from ED with ESC-X, and rerun the program with

```
>1 director showpix2
```

Note that 10 is a line number, and the syntax is blit buffer, fromx, fromy, tox, toy, width, height. A "square wipe" will appear in which Seascape at first fills the screen, but a small square window containing Moondance gradually expands from the center of the screen until Seascape is covered up. The reason is that line 10 sets width and height to a value of 2*dx, and the program steps dx from 0 to 100.

Animated Graphics and Movies

To make animated movies, purchase The Director Toolkit and follow the instructions given in the Enhanced Blit Utility and the MakeANIM documents. The instructions are detailed and will not be reviewed here. DPaint III, unlike its predecessor DPaint II, can generate sophisticated page-flipping movies. The Director is still necessary to link the user to the pictures and allow interactions by means of the mouse and keyboard; but unfortunately, since its current version predates DPaint III, it is not compatible with the DPaint page-flipping animations, so one must use the more awkward animation methods provided by The Director Toolbox. Doubtless a later release will fix this problem.

A Staircase Program for the Müller-Lyer Illusion

The next program, called Muller, is a staircase procedure for measuring the Müller-Lyer illusion. To speed up the experiments, the step size of the staircase is halved on each reversal in response, like a simplified version of the PEST procedure devised by Taylor and Creelman (1963). (This program is not a full implementation of PEST.) The illusion figure appears on the screen in the Brentano format, as a horizontal line with rightward pointing chevrons at each end, bisected (or approximately so) by a leftward pointing central chevron. The subject adjusts the position tox of this central chevron, moving it to the left and right by pressing keys 1 and 2. The step size dx halves on each response reversal. When the step size falls below 1 pixel, the central chevron disappears and the result of the trial (the value of tox) appears at top left of the screen, from where the experimenter writes it down. The subject hits any key and the next trial begins automatically. To exit from the program, click the left mouse button and hit any key.

The program, Muller, will be SAVEd as usual on the program disk in df1:. Note: in the CLI window, do NOT simply type

```
1>ed df1:Muller
```

because the current window is still df0:. Unless this is changed to df1:, after one EDits the new program, the computer will attempt to SAVE it to df0:, where the locked (write-protected) Director disk is residing. Instead, in the CLI window, the following should be typed:

```
1>cd df1:
1>ed Muller
```

This brings one into the ED window, with a message at the bottom "Creating a new program." Within ED, type

```
REM Muller: staircase for Müller-Lyer

LOAD 1,"df1:pix/blank4"
LOAD 2,"df1:pix/arrow"

10 MOVE 10,10: TEXT tox          :REM print results of previous trial
tox=100 + ?100                   :REM inner fins start randomly between 100 & 199
dx=32                            :REM initial step size
DISPLAY 1                        :REM display buffer 1
BLITDEST 1                       :REM send all pictures to buffer 1
TRANSPARENT 0                    :REM off
BLITDEST 2,0,50,0,50,320,130     :REM copy outer fins from buffer 2 to 1

20 GETKEY c
c=c-48                           :REM ASC(1)=49, ASC(2)=50
IF c=1: tox=tox-dx:ENDIF         :REM keys 1 & 2 move inner fins left & right
IF c=2: tox=tox+dx:ENDIF
BLIT 2,30,0,81,75,170,50         :REM repaint horizontal line from buffer 2 to 1
BLIT 2,0,0,tox,75,27,50          :REM repaint inner fins at position tox
IF c # oldc: dx=dx/2:ENDIF       :REM on each reversal, halve step size
IF dx < 1: GOTO 10:ENDIF         :REM exit to next trial
oldc=c                           :REM oldc detects response reversals
GOTO 20
```

(Note that the inequality sign in The Director is #. It is not< > as in some dialects of BASIC. Also, ?10 generates a random number between 0 and 9, and ?100 generates a random number between 0 and 99.)

The ends of the finned line, as shown in Figure 1, are at 80 and 280 pixels on the screen, so its center is at 180 pixels. However, most subjects set the subjective center near 165 pixels, so that a line of 85 pixels with outward fins looks the same length as a line of 115 pixels with inward fins—an illusion of 35%.

It is often a nuisance to have to copy results from the monitor screen during an experiment. The Director has no commands to allow one to print directly, but the next best thing is to write the results into a text file on disk during the experiment and then print them out afterwards. This is described later.

Control of Timing

Many experiments require precise control of timing. For example, one may wish to display a picture for a particular number of frames. If the picture in buffer 1 is to be displayed for exactly one frame (16.67 msec), followed by a blank frame already loaded into buffer 2, this can be accomplished reliably with the commands

```
DISPLAY 1                        :REM desired picture
DISPLAY 2                        :REM blank
```

This always exposes the picture for exactly one frame (16.67 msec).

Problems arise when longer presentations are required. The DISPLAY command can be repeated in a FOR loop, thus:

```
FOR i=1 TO nframes
    DISPLAY 1                    :REM desired picture
NEXT
DISPLAY 2                        :REM blank
```

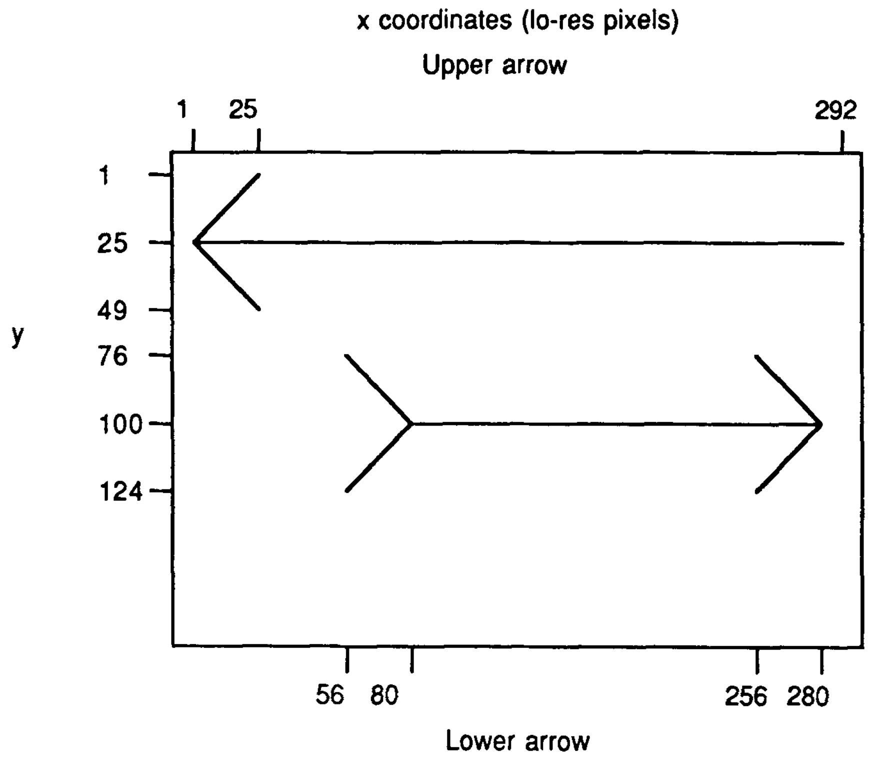

Figure 1. Load DeLuxePaint II (or III) and select lo-res screen format and a four-color palette. Turn on Coords from the Preferences menu; then draw the picture in white on a black background with the exact dimensions shown. Then SAVE the picture to disk under the name arrow. This picture is used in the Muller program.

However, this program will sometimes miss a screen synchronization pulse and display the picture for one or two frames longer than desired. Control of long display times can be made somewhat more reliable, although still far from perfect, with the program fragment below:

```
DISPLAY 1                   :REM desired picture
FOR i=1 TO nframes
    PALETTE-1,0             :REM-1=current buffer's palette. 0=not permanent.
NEXT
DISPLAY 2                   :REM blank
```

This works to control the display timing because the palette command can only be performed when the screen is updated for a new frame. We measured the picture durations with a Global Specialities 5001 digital timer inserted into the R, G, or B lead between the computer and the monitor, and found that when nframes was set to 10 in the FOR loop above, to obtain a desired display time of 10 frames, the picture was actually displayed for 10 frames about 80% of the time. For the other 20% of the time, the picture was displayed for 11 frames. Thus, we do not know of a really accurate way to display pictures for longer than a single frame, because mis-synchronization errors can occur. The PALETTE command works better than the PAUSE command, because the latter has a time resolution of .02 sec and is not synchronized to the updating of the display screen. BLIT commands should be used with caution where precise timing is required, because BLIT operations do not wait for a new frame before they begin.

The PALETTE command can also be used as a rudimentary tool for measuring reaction times, as is illustrated by the following program fragment:

```
DISPLAY 1
FOR counter=0 TO nframes
    IFMOUSE xloc, yloc
    IF xloc>1:GOTO 5:ENDIF    :REM xloc=1 until mouse is clicked
    PALETTE-1,0
NEXT
Rtime=17* counter             :REM estimate of reaction time
```

This example makes it clear that reaction times can only be measured to an accuracy of about ±17 msec. Fortunately, the IFMOUSE command in the above loop (and the related IFKEY command) does not alter the display timing. Thus, if one were to use either color cycling or page flipping to set a displayed picture in motion, the smooth motion would not be degraded by introducing an IFMOUSE or IFKEY command to wait for a subject's response.

Collecting Keyboard Responses

In most experiments, pictures are presented on the Amiga screen and the subject's keypress responses are collected. The Director is primarily a graphics program, and its string handling abilities are limited; it provides only a single string array into which keyboard responses must be put. This section shows how to put keypresses into the array. For further information, see pages 4-4 to 4-5 and 6-7 to 6-9 in The Director manual.

The ARRAY must always be declared before it is used, specifying the maximum number of characters it will hold and size of each character. For alphabetical letters or numbers from 0 to 255, the size is 1, and for numbers from −32767 to +32767, the size is 2. The larger the number and size of characters declared, the more memory is taken up.

In the following program, the user INPUTS two words (a maximum of 10 characters each) that are put into the array letter by letter, one letter to a cell. The first word starts at $(0), the 0th cell of the array, and the second word starts at $(10), the 10th cell. The command TEXT $(10), 8 prints on the screen the whole 8-letter string starting at the 10th cell and ending at the next empty cell in the array. If the word in the array is longer than 8 letters, it is truncated after 8 letters. If it is shorter, the whole word is printed. If the word actually began at $(9), the first letter will be truncated off the word. The user can also interrogate the contents of the array with the command TEXT @(10), which prints out the ASCII code for just the character in the 10th cell, not the whole string.

Suppose the array contained:

```
Cell No.   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Contents   F o r e v e r       A  m  b  e  r      y  e  s
```

the command TEXT $(10),8 would return

```
Amber
```

The letters ye from yes lie within 8 letters of $(10), but they would not be printed, because cell 15 is empty. The command TEXT @(10) would return

```
65
```

which is the ASCII code for the A in the 10th cell. Remember that the ASCII code for A is 65, for a is 97, and for 1 is 49.

The program is as follows:

```
REM fillarray

ARRAY 50, 1                      :REM you must declare the array before you use it
LOAD 1,"df1:pix/blank32"
MOVE 10,10
INPUT $($0),10                   :REM type in first word of 10 characters or less
MOVE 100,10
INPUT $(10),10                   :REM type in second word of 10 characters or less

MOVE 10,20
TEXT "$(i)     @(i)"

FOR i=0 to 15
    MOVE 10,10*i+30: TEXT $(i)
    MOVE 100,10*i+30: TEXT @(i)
NEXT i

MOVE 120,120
TEXT "Hit any key to exit"
GETKEY k
```

To run the program, type into the CLI window

```
    1>director fillarray
```

A black screen (actually blank32) will appear with a prompt at the top. Type in:

```
 _abacus<CR>                         _calculate<CR>
```

and on the screen will be seen:

```
 $(i)                                @(i)
 abacus                              97
 bacus                               98
 acus                                97
 cus                                 99
 us                                  117
 s                                   115
                                     0
                                     0
                                     0
                                     0
 calculate                           99
 alculate                            97
 lculate                             108
 culate                              99
 ulate                               117
 late                                108  Hit any key to exit
```

SAVEing Keyboard Responses to Disk

It is often convenient to SAVE keyboard data to disk files for later analysis.

Single keypresses. The following program collects six individual keypresses from the keyboard, puts them into the string array, and then prints them out on the screen:

```
REM saveinput

 ARRAY 20,1                          :REM declare the array before using it
 LOAD 1,"df1:pix/blank32"
 FOR i=0 TO 5
     GETKEY c                        :REM wait for keypress
     @(i)=c                          :REM insert keypress into cell (i) of the string array
 NEXT i
 MOVE 10, 30                         :REM position the cursor at x=10, y=30 pixels
 TEXT $(0)                           :REM print the whole string on the blank32 screen
 PAUSE 25                            :REM 25=duration of pause
```

The experimenter could now copy the data off the screen into his notebook. If one wants to SAVE the keypress data to a disk file, here called jfile, for later analysis, just add the following lines to the program:

```
 v=1                                 :REM v=0 to read a file, v=1 to write to it, v=2 to
                                      read and write
 OPEN v, "jfile"                     :REM note quotation marks around filename
 WRITE $(0)                          :REM write the whole string into a file called jfile
 CLOSE "jfile"
```

The following program will open the file just created, read its contents, and print them on the screen:

```
 REM readinput
 ARRAY 20, 1
 LOAD 1,"df1:pix/blank32"

 v=0                                 :REM v=0 means read only
 OPEN v, "jfile"
 READ v, $(0), 20                    :REM read the first line of text in the file and put the
                                     :REM first 20 characters into the string array
```

```
IF v=-1
   MOVE 10,50: TEXT  "End of file"  :REM v=-1 indicates error, e.g. if file does not exist
ENDIF                               :REM ENDIF must be one word
CLOSE "jfile"
MOVE 60,30                          :REM position the cursor at x=60, y=30 pixels
TEXT $(0)                           :REM print < =20 characters in the string array on to
                                    :REM the blank32 screen
PAUSE 25                            :REM 25=duration of pause
```

SAVEing to disk some words or numbers that are INPUT from the keyboard. It is easiest to SAVE records (strings) of fixed length; this wastes a bit of space by storing blank characters but makes it easier to keep track of where the words are. The next program accepts up to 25 words of maximum length 3 characters each (e.g., numbers of 1, 2, or 3 digits), which the user types into the keyboard, pressing <CR> (= carriage return) after each word. To exit after typing in a series of numbers, <CR> should be pushed; however, the space bar should *not* be hit before pressing the final <CR>. The data are SAVEd into a disk file, here called "numbers":

```
REM savenumbers
ARRAY 100,1
LOAD 1, "dfl:pix/blank32"
v=1                                 :REM write to file
OPEN v, "numbers"                   :REM note quotes around filename
5 FOR i=0 TO 100 STEP 4
6   MOVE 10, 10+2*i
7   INPUT $(i),3                    :REM user inputs number of 3 digits or less
    IF @(i)=0                       :REM exit if user presses <CR> (null input)
       GOTO 10
    ENDIF
    WRITE $(i)                      :REM write the number into the file
NEXT i

10 CLOSE "numbers"
MOVE 10, 180
TEXT "numbers completed"

REM now let's read the file we just wrote
v=0                                 :REM read from file
OPEN v, "numbers"
25 FOR i=0 TO 100 STEP 4
26  READ v, $(i), 4                 :REM not 3 ! Change it to 3 and see
    IF v=-1                         :REM exit at end of file
       GOTO 30
    ENDIF
27  MOVE 150, 10+2*i
    TEXT $(i)
NEXT i

30 CLOSE "numbers"
MOVE 160,180
TEXT "Hit any key..."
GETKEY dummy
```

Longer strings. In order to accept longer words up to a maximum of 10 characters instead of 3, make the following changes:
Delete lines 5, 6, and 7, and substitute

```
5 FOR i=0 TO 90 STEP 10
6  MOVE 10, 10+i
7  INPUT $(i), 10             :REM user inputs word of 10 characters or less
```

Delete lines 25, 26, and 27 and substitute:

```
25 FOR i=0 TO 90 STEP 10
26  READ v, $(i), 10
27  MOVE 150, 10+i
```

Analyzing data in files. The numbers in the "numbers" file (or any data file) are actually strings, which can be converted to numbers in The Director by the equals sign, as in

```
VALUE=$(5)
```

However, the best way to analyze numerical data is by quitting The Director and writing a separate program in BASIC. The following program reads the contents of the "numbers" file directly as single-precision numbers and calculates their mean and standard deviation. Note that this is the only program in this article that is written in BASIC; it is not a Director program.

```
REM mean&SD
OPEN "I", #1, "numbers"        :REM "I"=input=read from file
WHILE NOT EOF(1)               :REM the (1) in brackets refers to #1 in the OPEN command
    INPUT #1, n                :REM read the items
    sum=sum+n
    sumsq=sumsq+n*n
    num=num+1                  :REM count number of items
WEND
PRINT "Mean=";sum/num
SD=SQR(sumsq-sum)/(num-1)
PRINT "SD=";SD
```

To print the results on paper, the PRINT commands should be changed into LPRINT. Of course the printer should be properly connected and switched on.

Note two potential sources of error. If a trailing blank line were accidentally added at the end of the "numbers" file by hitting the space bar before the <CR> exit from the savenumbers program, the BASIC EOF (end of file) function will miscount the number of items in the file. If a "File not found" error message is obtained when one attempts to RUN mean&SD, it probably means that the "numbers" file is in a different drawer or subdirectory from the AmigaBASIC application.

Suppose the numbers file is in a directory called datafiles and the BASIC application is in a drawer called "Basic Stuff." From CLI, type

```
1>copy datafiles/numbers as BasicStuff/numbers
```

This will put a second copy of the numbers file into the Basicstuff subdirectory, leaving the original copy unaffected. ReRUN the BASIC program mean&SD.

Using Different Pictures on Different Runs of the Same Program

So far the names of pictures (such as "df1:pix/Seascape") have been a part of the program. But sometimes the user may want to run the same program with different pictures, which requires some way to input the picture names. For instance, one might need to run an experiment twice, using the same psychophysical procedure but a different set of three pictures on each run. Three steps are involved.

1. Go into DPaint and paint six pictures (2 runs × three pictures per run). SAVE them to the pix subdirectory on a disk in the external drive (df1:), naming the pictures at will. For present purposes, we shall use the pictures already painted called surround, mask, and target for the first run, and the demonstration pictures Seascape, Moondance, and Starflight for the second run.

2. Use ED to type in the names of the DPaint pictures just drawn (or selected), and SAVE them into a text file called (say) picfile1. This file will contain the *names* of the pictures (not the pictures themselves) that will be used on the first run of the experiment. Use ED again to enter the names of the pictures for the second run, and call the file picfile2.

3. The Director program given below called readnames is modified from page 6-7 of the manual. It will prompt the user to type in the name of one of the picture-name files just generated (e.g., picfile1). The readnames program will then LOAD the pictures named in picfile1 into screen buffers and display them in sequence.

To type in the names of the disk pictures, enter CLI and type

```
1>cd df1:
1>ed picfile1
```

Within ED, type in the program:

```
dfl:pix/surround<CR>
dfl:pix/mask<CR>
dfl:pix/target<CR>
end<CR>
```

(Reminder: <CR> means carriage return. No further reminder will be provided.) Note that the word "end" must be typed in. If not, or if the name of a nonexistent picture is typed, a "LOAD: Can't load:" error message will appear when one attempts to run the readnames program.

Exit to CLI. To type in the names for the second experimental run, type

```
1>ed picfile2
```

Within ED, type in the program:

```
dfl:pix/Seascape
dfl:pix/Starflight
dfl:pix/Moondance
end
```

The screen will return to CLI, and now that the two text files picfile1 and picfile2 contain the picture names, it is time to type in the program readnames, which resembles the earlier program readinput, and was modified from page 6-7 of the manual. Type

```
1>ed readnames
```

From within ED, type

```
REM readnames

ARRAY 200,1
LOAD 1, "dfl:pix/blank4"        :REM change this to "dfl:pix/blank32" for 32-color pictures
buff=1
mode=0
MOVE 10,10: TEXT "Picture-names are in textfile:"
MOVE 150,10: INPUT $(0),10  :REM enter name of textfile (previously generated by savenames)
OPEN mode, $(0)
IF mode=0
    MOVE 10,150: TEXT "Can't open file":END
ENDIF
10 READ EOF,$(0), 25
IF EOF=-1: wait=10:GOTO 30: ENDIF
COMPARE eq, "end", $(0)      :REM look for the word "end" in picfile
IF eq: GOTO 20: ENDIF
LOAD buff, $(0)
buff=buff+1
GOTO 10

REM got end, let's get parameter
20 READ EOF, $(0), 25
IF EOF=-1: wait=10: GOTO 30: ENDIF
wait=$(0)

REM now that we've got them all, let's flip pages
30 FOR i=1 TO buff-1
    DISPLAY i
    PAUSE wait
NEXT i
GOTO 30
```

Exit from ED by typing ESC-x <CR> and run the readnames program from within CLI by typing

```
1> director readnames
```

Upon seeing the prompt "Picture names are in textfile:", type

```
picfile1
```

The program will LOAD the three pictures' names in picfile1—namely, surround, mask, and target—and will flash them up in sequence. The disk drive light will go on as the three pictures are LOADed, and then the screen will be filled successively with gratings that slant one way, the other way, and zigzag.

Exit from the program by clicking the left mouse button and hitting <CR>, and rerun the readnames program by typing

```
director readnames
```

Upon seeing the prompt "Picture names are in textfile:", type

```
picfile2
```

and this time the program will crash. The reason is that all the pictures within a program must have the same number of colors in their palette. The palettes can contain 4, 8, 16, or 32 colors, but they must all have the same number. Each picture can, and usually does, have its own independent palette—one picture may have all earth tones, another may have metallic colors, and so on. But the second line of the readnames program

```
LOAD 1, "df1:pix/blank4"      :REM change this to "df1:pix/blank32" for 32-color pictures
```

loads in a 4-color picture. Since the demonstration pictures in picfile2 all contain 32-color palettes, this line must be changed to

```
LOAD 1, "df1:pix/blank32"
```

This will do the trick. Now type

```
director readnames
```

Upon seeing the prompt "Picture names are in textfile:", type

```
picfile2
```

and the program will flash Seascape, Starflight, and Moondance in sequence. Incidentally, it is not necessary to have exactly three pictures within each picfile. The readnames program keeps loading the pictures named until it encounters the word "end" in the picfile. One can experiment with a new file called picfile3 containing any number of 32-color pictures that one happens to have on the program disk.

In practice, the user would modify readnames to make it the front end of a complete psychophysical experiment.

Collecting Mouse Inputs

The mouse position (*x*,*y*) can be read with Getmouse, which waits for a mouse click. The following exercise uses the mouse to reset individual colors in the palette. To exit from the program, one moves the mouse cursor to the top left corner of the picture and clicks.

```
REM mousepalette
LOAD 1,"df1:pix/Seascape"
10 GETMOUSE x,y                          :REM wait for mouseclick and read x,y
IF (x=0) & (y=0)                         :REM to exit click mouse when at top left corner
                                          of screen
    GOTO 20                              :REM note syntax of IF
ENDIF                                    :REM endif must be one word
REM next two lines convert mouse x-coordinate (range 0 to 319) into gray level (0=black, 15=white)
REM and mouse y-coordinate (range 0 to 199) into palette colornumber (lowest color=0, highest=31)
colnumber=x*31/310                       :REM x chooses palette's colornumber from 0 to
                                          31
gray=(200-y)*15/200                      :REM x sets gray level from 0 to 15
buff=1                                   :REM buffer
perm=0                                   :REM permanence=0 changes picture, =1 changes
                                          palette too
COLOR buff, perm,colnumber,gray,gray,gray :REM gray,gray,gray go into the red, green and
                                         :REM blue components of the color palette
GOTO 10
```

Think of the picture as being divided into 32 invisible vertical columns, each representing a color in the palette. A click anywhere in that column changes the palette color to a shade of gray—light if the mouse cursor is at the top of the screen, dark if it is near the bottom. For example, clicking on the center of the little house on the skyline changes some rocks on the left to light gray. Clicking on the right-hand part of the house selects a different palette color and changes the flowers at the bottom to the same shade of gray. Clicking on the left hand margin changes the surround, making it light gray if one clicks near the top of the screen, black if one clicks at the bottom. Clicking on the top left hand corner will permit exit from the program.

Controlling a Pointer on the Screen from the Keyboard

It is often useful to have a radial line centered on the screen, which the subject can rotate to match the perceived orientation of a target line or the perceived direction of a movement, etc. The following program puts the orientation of such a line under the control of the keyboard. Since The Director in its original form does not support SIN and COS (although the Toolbox does), the pointer tip traces out a diamond instead of a true circle.

```
REM Keypointer

5 ABORT 1                               :REM click mouse then hit any key to abort
                                         program. 5 is a line number

LOAD 1,"df1:pix/blank32"

10 GETKEY k                             :REM waits for keypress. Ifkey would not wait.
                                         10 is a line number
                                        :REM k takes ASCII value of key pressed
x=x+k-53                                :REM ASC(5)=53, so x=0 if k=5, and x>0 if
                                         k>5

IF x<0: x=x+319: ENDIF
IF x>319: x=x-319: ENDIF
pen 1,0: move 160,100: draw xp,yp       :REM erase pointer
xp=x+80                                 :REM convert mouse x position to
IF x>160: xp=400-x: ENDIF               :REM xp,yp pointer co-ordinates
yp=100-x
IF x>80: yp=x-60: ENDIF
IF x>240: yp=420-x: ENDIF
PEN 1,1: MOVE 160,100: DRAW xp,yp       :REM draw the pointer
GOTO 10
```

This program provides fine variable-speed control of the pointer from the numeric keypad. Holding down keys 6, 7, 8, or 9 will rotate the pointer at increasing speeds clockwise. Holding down keys 4, 3, 2, or 1 will move the pointer at increasing speeds anticlockwise. Holding down key 5 will have no effect. Pressing the spacebar, or any letter key, will move the pointer in large anticlockwise or clockwise jumps, respectively.

Note that a similar rotating pointer can be drawn in DPaint without invoking The Director at all. Select a single-point pen and the straight line tool in DPaint. Select Coords from the Preferences menu (or press shift-backslash, just below the F10 key). Draw a line starting near the center of the screen and swing the line around in an arc without releasing the mouse button. The radius and angle of the line will appear in the menu bar. To print down the line without superimposing it on any picture on the screen, J should be pressed before releasing the mouse button. Releasing the button results in the line's being drawn on the spare screen page, where it may be analyzed later.

Aspect.ratio: A Staircase Program

This program for collecting experimental data by a staircase procedure measures peripheral acuity for textured stimuli, and it uses the three textures that were drawn earlier in DPaint, called surround, target, and mask, which consist of left-oblique stripes, right-oblique stripes, and a random pattern of zigzag stripes. First, the names of these textures should be placed into a short names file to be called tex1. The front end of the program contains a version of readnames which reads the tex1 file. From CLI, type

```
1>ed tex1
```

Within the ED window, type:

```
df1:pix/surround
df1:pix/target
df1:pix/surround
df1:pix/blank4
df1:pix/mask
df1:pix/spot
end
```

(The word "end" must be included. REM statements should not be placed into a text file.)
Run the program in the usual way with

```
1>cd df0:
1>director df1:Aspect.ratio
```

The program will prompt the user for the filename, so be sure to type df1:tex1 not just tex1. The disk drive (df1:) must be included; otherwise the program will wrongly search the current directory, which is df0:, and fail to find tex1. An area of (say) 1,024 and an eccentricity of (say) 100 should be typed. The program will load the named pictures and cut and paste them to present horizontal or vertical bars of striped texture in the peripheral visual field. Each time the target is presented, the subject judges its aspect ratio, hitting key 1 on the keyboard if the target looks taller than it is wide (a vertical bar) and key 2 if it looks wider than it is tall (a horizontal bar). If the subject makes the correct response, the task is made harder for the next trial by making the target more square, and if the subject makes an incorrect response, the task is made easier by making the target more elongated. After a preset number of reversals, the results are printed out on the screen.

The program embodies a number of the techniques described in this article. The reader is encouraged to use this program as a kind of pie shell to hold the filling of his or her choice. The program is as follows:

```
REM aspect.ratio
REM Peripheral acuity for textured stimuli, measured by a staircase
REM User presets area and eccentricity (=screen position)
REM Hit key 1 if bar looks horizontal, 2 if it looks vertical.
REM Keys 1 and 2 vary the target's aspect ratio. Key 9 aborts the program.

ARRAY 200,1                                          :REM array will hold inputs
LOAD 4, "df1:pix/blank4"
DISPLAY 4
MOVE 10,10: TEXT "Picture names file? (=df1:tex1)"
MOVE 10,50: INPUT $(0),10                            :REM enter e.g. df1:tex1
REM 1=surround, 2=target, 3=surround, 4=blank, 5=mask, 6=spot

MOVE 10,70: TEXT "Loading...."
OPEN mode, $(0)                                      :REM open file containing names
                                                      of pictures
IF mode=0
   MOVE 10,150: TEXT "Can't open file": END
ENDIF
2  READ EOF, $(0), 25
IF EOF=-1: wait=10: GOTO 3: ENDIF
COMPARE eq, "end",$(0)                               :REM end of file?
IF eq: GOTO 30: ENDIF
MOVE 10,90: TEXT $(0);"     "
REM print names of files that are loading. Spaces obliterate garbage.
LOAD buff, $(0)                                      :REM load pictures into buffers
buff=buff+1
GOTO 2

3  DISPLAY 4                                         :REM display buffer 4 (=blank)
GOSUB 110                                            :REM collect inputs from user
4  DISPLAY 1                                         :REM surround texture
TRANSPARENT 0                                        :REM turn off transparency
CYCLE 0                                              :REM turn off color cycling
yc=100                                               :REM tvertical position of target
dx=8                                                 :REM countdown 8 reversals
```

```
5  width=area/height
IF ?2=1:temp=height:height=width:width=temp:temp:ENDIF  :REM randomly start skinny or
                                                         squat

REM display stimuli

10 fromx=?200:fromy=?150                                :REM copy random piece of
                                                         texture into stimulus
tox=xc-width/2: toy=yc-height/2
DISPLAY 5                                               :REM mask
PAUSE 1
BLITDEST 1                                              :REM send target to surround
                                                         texture in buffer 1
BLIT 2,fromx,fromy,tox,toy,width,height                 :REM stimulus patch
GETKEY c                                                :REM wait for keypress
BLIT 1,0,0,tox,toy,width,height                         :REM blank out target with piece
                                                         of surround

REM keypress alters aspect ratio

c=c-48                                                  :REM convert ACSII code back
                                                         to number: ASC(1)=49
IF c=1:GOSUB 22:ENDIF                                   :REM subject hits horizontal key
IF c=2:GOSUB 24:ENDIF                                   :REM subject hits vertical key
IF c=9:GOTO 30:ENDIF                                    :REM key 9 aborts to printout
GOTO 5                                                  :REM next stimulus

REM horizontal key 1 pressed
22 IF height>width
    height=height-2:crect=0                             :REM wrong
    ELSE height=height+2:crect=1                        :REM right
ENDIF
RETURN

REM vertical key 2 pressed
24 IF height>width
    height=height+2:crect=1
    ELSE height=height-2:crect=0
ENDIF
RETURN

REM printout

30 DISPLAY 4                                            :REM blank
MOVE 10,10: TEXT "Eccentricity (xc)=";xc
MOVE 10,30: TEXT "Height=";height
MOVE 10,50: TEXT "Width=";width
MOVE 10,70: TEXT "Area=";area
MOVE 10,90: TEXT "Click left mouse button to exit"
MOVE 10,110: TEXT "Hit '1' for another run with same stimuli"
MOVE 10,130: TEXT "Hit any other key to reset stimuli"
GETKEY k
CLEAR
IF k=49:GOTO 4:ENDIF                                    :REM another run with same
                                                         stimuli
GOTO 3                                                  :REM reset stimuli

REM collect inputs from user
110 MOVE 10,10: TEXT "Area (256-1024)";
INPUT $(0), 4                                           :REM put input into array
area=$(0)
width=16*(?2+1)* (?2+1)                                 :REM randomly set to 16, 32, or
                                                         64
height=area/width
MOVE 10,30: TEXT "Eccentricity (50-250)";
INPUT $(0), 3                                           :REM put input into array
xc=$(0)
MOVE 10,150: TEXT "Hit any key for first trial";
```

```
GETKEY dummy
CLEAR
RETURN
```

This program can be used to explore peripheral acuity for different textures. The user can paint a different set of textures in DPaint and call them (say) surround2, target2, and mask2. One can compare the acuity for random dots against a surround of horizontal dashes, or for fine texture against a surround of coarse texture, and so on. Within the ED window, type a names file called text2:

```
df1:pix/surround2
df1:pix/target2
df1:pix/surround2
df1:pix/blank4
df1:pix/mask2
df1:pix/spot
end
```

On running the aspect.ratio program, type in tex2 instead of tex1 at the prompt.

REFERENCES

Anstis, S. [M.] (1986). Visual stimuli on the Commodore Amiga: A tutorial. *Behavior Research Methods, Instruments, & Computers*, **18**, 535-541.

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

Commodore-Amiga Inc. (1988). *The Amiga DOS manual* (2nd ed.). New York: Bantam Books.

Levy, S. (Ed.) (1986). *Compute!'s Amiga programmer's guide*. Greensboro, NC: Compute! Publications.

Taylor, M. M., & Creelman, C. D. (1963). PEST: Efficient estimates on probability functions. *Journal of the Acoustical Society of America*, **41**, 782-787.

NOTE

1. The reason is that a picture with a standard 32 colors needs 5 bitplanes ($2^5=32$) but with a 4-color palette needs only 2 bitplanes ($2^2=4$). A single lo-res plane with two colors at 320×200 pixels requires 8,000 bytes of random-access memory (RAM), so one could get close to 50 frames into 512K of memory. On the other hand, a 640×400 32-color display requires 256,000 bytes, so, allowing some room for the program, one can get just one frame into 512K of memory. This rules out any animation. For this reason, all DPaint pictures in this article are low-resolution (lo-res), namely 320×200 pixels.

APPENDIX
Some Useful ED (Screen Editor) Commands

Immediate commands (Hold down <Ctrl> and press key):

Ctrl-A	Insert line at cursor
Ctrl-B	Delete current line
Ctrl-D	Scroll text downward
Ctrl-E	Move cursor to top or bottom of screen
Ctrl-N	Delete character at cursor
Ctrl-O	Delete word or series of spaces
Ctrl-U	Scroll text upward
Ctrl-Y	Delete to end of current line

Extended commands (Precede by pressing and releasing <Esc>):

B	Move cursor to bottom of file
E/*string1*/*string2*/	Exchange *string1* with *string2*
EQ/*string1*/*string2*	Exchange, but query first
F/*string*/	Find current string
J	Join current line with next line
Q	Quit without saving text
T	Move cursor to top-of-file
X	Exit, save text

(Manuscript received August 28, 1989;
accepted for publication August 29, 1989.)