PYRAMIDE

GRAPHIC TOOLKIT

by Mick Andon

GRAPHIC TOOLKIT

by Mick Andon

INTRODUCTION

This toolkit provides a range of functions and procedures aimed at exploiting the screen and graphic capabilities of the QL. The extensions (occupying around 11 K of code) are accompanied by several applications and example routines. Most of these are described in detail in this manual. An alphabetical index for the extensions can be found at the back.

Before using GRAPHIC TOOLKIT, you should make a backup as follows:

> Place GRAPHIC TOOLKIT in mdv2_
> Place a BLANK cartridge in mdv1_
> Type: **LRUN mdv2_backup**

To install the extensions . . .

either : Reset the QL, insert GRAPHIC TOOLKIT in mdv1_ and press F1 or F2

or : Reserve memory for the extensions. - addr=RESPR (11000) : CALL addr

or : Type - LRUN mdv1_boot

Syntax of command descriptions etc.

In this manual, the syntax for descriptions conforms to these rules :

UPPER CASE words . . .	should be typed as written (in upper or lower case)
Lower case words . . .	are descriptive
[. . .]	indicates an optional parameter

Apart from the boot and backup programs, the filenames on the toolkit cartridge have the following descriptive suffixes :

"_bin"	indicates a binary file
"_bas"	is a BASIC program
"_scr"	is a 32 K screen file
"_set"	is a character set (font)
"_x"	is an 'EXEC'able file

POINTER x, y   (Procedure)
XPOINT   (Function)
YPOINT   (Function)
POINTKEY   (Function)

Places a pointer at position x,y which can be moved around the screen using the four
cursor keys, joystick or mouse. The pointer will remain on screen until either SPACE,
ENTER or ESCape (or the corresponding mouse or joystick button) is pressed .

On return from the procedure POINTER, three variables are set -
XPOINT   returns the last x co-ordinate of the pointer.
YPOINT   returns the last y co-ordinate of the pointer.
POINTKEY returns a value of 0 if the key pressed was 'ESC'
                             1 if the key pressed was 'SPACE'
                             2 if the key pressed was 'ENTER'

Note: Examine the routine "pointer_demo_bas" for an example of usage.

POINTBUF   (Function)

The pointer is defined on a matrix of 7 lines of 8 pixels (together with a mask of the
same dimensions). It may be redefined by POKEing new values into the pointer buffer
(the relevant address is returned by the function POINTBUF). The pointer is intended
principally for use in mode 4 - since in mode 8 the matrix becomes effectively 7x4
pixels - however it is possible to define pointers for use in mode 8 (attention must be
paid to avoid setting flash bits).

The data for the default arrow is as follows:

Arrow (binary)		hex	Mask (binary)		hex
11111110	00000000	FE00	00000001	11111111	01FF
10000010	00000000	8200	00000001	11111111	01FF
10001110	00000000	8E00	00000001	11111111	01FF
10100110	00000000	A600	00000001	11111111	01FF
10110011	00000000	B300	00000000	11111111	00FF
11111001	00000000	F900	00000000	11111111	00FF
00001111	00000000	0F00	11110000	11111111	F0FF

Note: Each line is defined by the high byte of a word (all the bits in the low byte are
         reset in the case of the shape and set in the case of the mask).

Example (mode 8 pointer):

```
100  RESTORE 120
110  FOR x=0 to 13 : READ num$ : POKE_W POINTBUF + (2*x), DEC(num$)
120  DATA "AA00", "8000", "8000", "B200", "8000", "8800", "0200" : REMark  arrow
130  DATA "00FF", "00FF", "00FF", "00FF", "00FF", "00FF", "FCFF" : REMark  mask
```

POINTSPD value

POINTSPD sets the pointer speed. Ten speeds are available, from 1 (slowest speed) to 10. On installation POINTSPD is set at 5. The pointer speed will remain unchanged until a new value is set by POINTSPD.

FRAME w, h, x, y, scr_x, scr_y, virtual_screen_addr

FRAME sets up a "window" (w pixels by h lines) at co-ordinates x,y and displays "behind" that window, a section of a virtual 512x256 screen whose origin is located at virtual_screen_addr. The virtual screen may consist of a 512x256 image that has been stored in memory, or it may take its origin at any memory address - even within the current screen display (131072 to 163840).

The third and fourth co-ordinates indicate the point (measured relative to the virtual screen's origin) that will coincide with the display window's origin.

All horizontal co-ordinates are rounded down to coincide with long word addresses. This is in order to increase speed of execution in shifting large sections of screen memory. The virtual screen start address may be anywhere in RAM, however, it is important that the window defined by w,h,x,y falls within the normal 512x256 screen display or the QL will probably crash.

Example: 100   SETWIN 4 : LIST#0 : LIST#1 : LIST
           110   RECT#2, 128+SMODE/2, 82, 45, 22, 0
           120   w=128 : h=80 : x=48 : y=24 : scrx=48 : scry=24 : addr=131072
           130   REPeat loop
           140   IF KEYROW(1)&&2 : scrx=scrx-16
           150   IF KEYROW(1)&&16 : scrx=scrx+16
           160   IF KEYROW(1)&&4 : scry=scry-8
           170   IF KEYROW(1)&&128 : scry=scry+8
           180   FRAME w, h, x, y, scrx, scry, addr
           190   END REPeat loop

Note: The FRAME routine is used by the program "transfer_x" to pan/scroll the second screen within a window.

PAINT x, y, colour, [source_colour]

PAINT will paint an area of uniform colour (where x,y is a point inside that area), using another colour (0 to 255). The two co-ordinates, x and y, should be given as absolute (based on a screen of 512x256) for both mode 4 and mode 8. If the paint colour specified is the same as the original colour found at x,y then the paint will not take place.

A fourth parameter may be specified (source_colour) - in which case the paint will only execute if the colour found at x,y has the same value as that parameter.

PAINT contd.

To fill more complex shapes, more than one PAINT will be required. However, no
memory is used by the routine for storing co-ordinates etc., and execution is
reasonably rapid. The procedure POINTER is an ideal means of positioning x and y
when using mode 4.

Example: 100   INPUT#0,"Colour ? "; colour : NOKEY : POINTER 250, 120
             110   PAINT XPOINT, YPOINT, colour
             120   POINTER XPOINT, YPOINT : IF POINTKEY=2 : GOTO 100
             130   GOTO 110

ALCHP (nbytes)   (Function)
RECHP address   (Procedure)

ALCHP is used to allocate memory space (in the Common Heap) for use by machine
code programs, storage of screens etc. Its use is similar to that of the function RESPR
(which reserves space in the Resident Procedure area), the main difference being that
the memory may be later released to the Common Heap when no longer required.

Example: addr = ALCHP (32768)   . . . will reserve 32768 bytes

Since 32768 bytes is the screen memory size, you could for instance load a screen
directly from microdrive into memory using **LBYTES mdv1_name_scr,addr** and then
place the image on the screen using **FRAME** or **PLACE**.

RECHP is used to release memory allocated by the above. It is important to save the
base address (addr - returned by the function ALCHP) if you wish to use the procedure.
Type RECHP (followed by the address) to release the memory to the Common Heap.

Important note: The Common Heap may become fragmented by multiple
                   allocations . . .eg: PRINT FREE
                                            q = ALCHP (40000) : PRINT FREE
                                            a = ALCHP (512) : PRINT FREE
                                            RECHP q
                                            PRINT FREE

The RECHP appears not to have worked - this is because the small space (which is
reserved by "a") is separating two sides of the Heap area. If you now type - **RECHP a,**
although you are only releasing 512 bytes, typing FREE will reveal that the heap has
been restored to one area. Using a directory device (eg: mdv2_) for the first time, will
set up a "definition block" in the Common Heap in the same way as the example
above.

It is advisable to previously access all the directory devices likely to be used during
large allocations of the Common Heap.

FREE

This function will return the approximate number of bytes free for use by Superbasic (the largest single free space), and is essential as a means of keeping check on memory usage.

Example: 100   DEFine PROCedure pr
              110   v = v+1
              120   AT 0,0 : PRINT FREE; " Bytes left " : pr
              130   END DEFine
              140   CLEAR : v=1 : pr

After a while the "Out of memory" message will appear. It is often difficult to locate a cause of memory wastage in large programs. Using FREE (within a main loop) as follows could help to do so . . .   IF KEYROW(3) = 16 : AT#0,0,0 : PRINT#0, FREE

BVAR

Unlike the QDOS System Variables (located above the screen at $28000), the position in memory of the Basic Variables is not fixed. The BVAR function returns the base address of this area, thus giving access to the variables which are at fixed offsets from that address.

QDOS$

This is the QDOS equivalent of the basic function VER$ and will return the QDOS version number for you QL (ie: 1.03).

STAMP mdv, n1, n2
CHECK mdv, n1, n2

These are used for anti-copy protection. To invisibly mark a **previously formatted but blank cartridge** type STAMP · followed by 3 parameters · microdrive number (1 or 2), then two numbers (each between 2 and 200). You must then save a program on the cartridge before removing it from the drive.

To test for the two numbers, type · CHECK · followed by the microdrive number and the two 'pass' numbers. After reading the cartridge, if the numbers are correct, nothing will happen · otherwise the error "Bad or changed medium" will occur.

This routine is very effective if incorporated within a Supercharge compiled program to check for a 'master' or 'original' cartridge in one of the drives.

Example: 100   CHECK 2, 48, 58 : REMark the program will stop if 48 & 49 are not found to be STAMPED on the cartridge in mdv2_

LOCK (Procedure)
UNLOCK (Procedure)
LOCKED (Function)

LOCK will dis-enable 'Break' caused by CTRL/SPACE as well as 'pause' normally caused by CTRL/F5. It is important **NOT to LOCK TWICE** otherwise the computer will crash (use the function LOCKED to check).

UNLOCK has the reverse effect, ie: it will re-enable CTRL/SPACE & CTRL/F5.
The function LOCKED returns a value of 1 if locked or 0 if not locked.

Example: 100 IF NOT LOCKED THEN LOCK : ELSE PRINT "Already locked !"

DEVSTAT ("device") (Function)

This function tests and will return the status of any directory device connected to the QL. It is a useful means of avoiding program crashes during directory or access from non-existant cartridges or disks etc. (the device name must be enclosed by inverted commas). The status number returned in the event of an error is the QDOS error code (a small negative value) - otherwise a zero is returned.

Example: 100 IF DEVSTAT ("mdv2_") < 0 : BEEP 3000,50 : Print "Place a cartridge
in mdv2_ & press a key" : PAUSE : CLS : GOTO 100
110 PRINT "..................... OK"

REPORT [#n], error_number (Procedure)

REPORT will print the equivalent QDOS error message to the given channel (the default is #0). Error codes are between -1 and -21.

Example: REPORT#2,DEVSTAT("flp1_")
. . . will print "Not found" in window #2 if a disk drive is not connected or there is no disk in the drive.

RESET (Procedure)

This is a software equivalent of pressing the reset button on the side of the QL.

WAIT value (Procedure)

Suspends a program for a period depending on the value given. Unlike the PAUSE command, WAIT is unaffected by use of the keyboard. In addition, the length of wait is unaffected by compilation using Supercharge - (a value of 200 gives around 1 second, 1000 around 5 seconds, 2000 around 10 seconds etc.).

SCADDR (x, y) (Function)

Returns the address in the screen memory indicated by co-ordinates x and y. The co-ordinates must be given as absolute (512x256 pixels). This procedure is useful for locating start and end addresses when saving just a section of screen.

```
Example:   100   REMark to save screen from 0,22 to 512,200
           110   start = SCADDR (0,22)
           120   finish = SCADDR (512,200)
           130   length = finish - start
           140   SBYTES mdv1_screen_section, start, length
```

POKE_S x, y, value (Procedure)
PEEK_S (x, y) (Function)

POKE_S pokes a word length value to the screen at the given co-ordinates while PEEK_S reads the screen at x,y. An example of application is to stock values using the screen as memory (either to save on RAM elsewhere or in order to read values from "data boxes" on screen for menu selection using a pointer).

DISP n (Procedure)

Switches the QL screen display according to the value given (between 0 and 4). The corresponding effects are as follows:

```
    0   -   switches off the display
    1   -   sets the 4 colour display
    2   -   sets the 8 colour display
    3   -   sets the 2nd screen 4 colour display
    4   -   sets the 2nd screen 8 colour display
```

Changing between the displays does not affect the screen RAM - only the way in which it is interpreted. Switching is instantaneous.

```
Example:   100   DISP 0
           110   LBYTES mdv1_screen_scr, 131072
           120   DISP 1
```

SMODE (Function)

SMODE returns a value of either 4 or 8, depending on the QL's current mode. It is useful for setting variables that require different values for mode 4 or mode 8, or checking that the correct mode is set without re-calling the MODE command.
ie: IF SMODE < > 4 THEN MODE 4

SETWIN n

This command selects one of six screen layouts for the three windows opened at switch-on. The first two are identical to those that appear on pressing respectively F1 or F2 after a reset (the screen mode is not changed by SETWIN). The third layout is similar to the second, but expanded to meet the edges of the screen. The fourth is the same as that produced by pressing F1, but #1 is narrower and #2 wider (which is more practical for displaying program listings). Display n°5 stacks the three windows from top to bottom and n°6 sets a full screen (512x256a0x0) for #1.

To select one of the above window layouts, type SETWIN followed by a number between 1 and 6. The INK/PAPER/STRIP colours and BORDERS are also set, and all windows cleared by this procedure.

FONT [#n], addr1, addr2

The procedure FONT sets the 2 character fonts for any screen or console channel. If no channel is specified, the default is #1. The two parameters that follow are the base addresses for font 1 (characters 31 to 127) and font 2 (characters 127 to 191). Either one or both can be set - giving a 0 as one of these two parameters (in place of an address) will set that font to the QL ROM set. A character font normally requires 875 bytes (reserved using RESPR or ALCHP).
The format for a character font is as follows:

1st byte	the lowest valid character value
2nd byte	the number of valid characters -1

. . . followed by 9 bytes of definition for each of the characters (see "definer_bas").

FONT1BASE [(#n)]
FONT2BASE [(#n)]

These functions return the base addresses for respectively the first and second fonts set for that given channel. If no new fonts have been installed then the ROM font addresses are returned. (The ROM character set addresses vary depending on the QL version).

CURSZ [#n], h, v

CURSZ sets the character spacing in horizontal and vertical increments for a given window (default #1). In mode 4 for instance, the default increments for CSIZE 0,0 are 6 and 10. (By using CURSZ 6,9 in mode 4, it is possible to increase the maximum number of character lines displayable by 10%).

Note: The commands NEW, MODE and CSIZE will cancel increments that have been set by CURSZ.

XPIXS (#n) (Function)
YPIXS (#n) (Function)
XCHRS (#n) (Function)
YCHRS (#n) (Function)

Typing - PRINT XPIXS(#n), YPIXS(#n) - where n is a screen or console channel, will print the window size in pixels (horizontally and vertically) of #n. This information can be used to set sizes/limits etc. for objects placed within a window area.

XCHRS and YCHRS are similar except that they return the window size in terms of characters, which will of course vary according to the character size set by CSIZE.

XPIXP (#n) (Function)
YPIXP (#n) (Function)
XCHRP (#n) (Function)
YCHRP (#n) (Function)

These functions return the current cursor position within the specified channel. XPIXP and YPIXP will return the x,y co-ordinates in pixels, and XCHRP and YCHRP will return x and y in character co-ordinates.

CHANID (#n) (Function)

Returns the QDOS channel identification for the given Basic channel ID. When converted to a hex string - eg: PRINT HEX$ (8, CHANID(#2)) - the QDOS channel ID consists of - in the low word (RHS) a reference to the location in the channel table, and in the high word, a "tag" that increments each time that channel is opened.

Example: (After switch-on) #2 would have a QDOS ID of $00020002

CURSEN #n (Procedure)
CURDIS #n (Procedure)

The procedure CURSEN (followed by a screen channel) will enable the cursor in that window. CURDIS has the reverse effect - the cursor is "unprinted" and dis-enabled for the given channel.

The function INKEY$ takes data from the keyboard without enabling the cursor. Using CURSEN it is possible to enable the cursor in a window so that when INKEY$ is called, the keyboard queue will be switched to that window and the cursor will flash.

Example: 100 CURSEN#2
            110 PRINT#2, "You have 10 seconds to press a key > > >"
            120 key$ = INKEY$(#2,500)
            130 CURDIS#2 : CLS#2,3

MOVXY	(Procedure)
XP	(Function)
YP	(Function)
SPA	(Function)

The procedure MOVXY has the effect of reading the four cursor keys (and SPACE), and incrementing/decrementing x and y values (which are returned as XP and YP respectively). MOVXY is subject to previously set limits (LIMXY) and increments (XYSTEP). It is capable of replacing lengthy cursor moving loops with just a few instructions.

Example: 100 SETWIN 6
110 REPeat loop : MOVXY : BLOCK 2, 1, XP, YP, 2 : END REPeat loop

---

SETXY x, y   (Procedure)

SETXY followed by two co-ordinates sets respectively XP and YP for use with the procedure MOVXY.

LIMXY left, up, right, down   (Procedure)

It is usually necessary to set a limit to the possible values of XP and YP. LIMXY is used for this purpose. The parameters are (in order) - lowest possible values for XP and YP, followed by the highest possible values for XP and YP. (When the extensions are first installed, LIMXY is set at 0, 0, 504, 250). It is important to make sure that the co-ordinates set in SETXY fall within the limits set by LIMXY.

XYSTEP xstep, ystep   (Procedure)

XYSTEP sets the increments in x and y for the procedure MOVXY. When one of the cursor keys is pressed, XP or YP will increase/decrease by the amount set in XYSTEP. (On installation of the extensions, both steps are set at 1).

Example: 100 IF SMODE < > 8 : THEN MODE 8
110 SETWIN 6 : CLS
120 LIMXY 0, 0, 240, 100 : XYSTEP 8, 4 : SETXY 0, 0
130 REPeat loop
140 MOVXY
150 IF SPA : colour=0 : ELSE : colour=RND(7) : END IF
160 BLOCK 16, 8, 252-XP, 128-YP, colour
170 BLOCK 16, 8, 252+XP, 128+YP, colour
180 BLOCK 16, 8, 252+XP, 128-YP, colour
190 BLOCK 16, 8, 252-XP, 128+YP, colour
200 END REPeat loop

CROSS [#n], x, y, colour   (Procedure)

CROSS prints a cross-wire cursor in a channel (default #1) at window position x,y using INK colour. It is subject to the value of OVER set for that window - (so if using OVER -1, printing the cursor a second time at the same position will have the effect of unprinting it).

RECT [#n], width, height, x, y, colour   (Procedure)

Prints a rectangle in a window (default #1) in the specified colour, where x and y are the pixel co-ordinates of the rectangle's top left corner. (RECT may be 'XOR'ed by setting OVER [#n], -1).

GRID [#n], [size]   (Procedure)

GRID will print a grid over the whole or part of the screen which **will remain displayed until SPACE is pressed**. It is intended for use in measuring proportions for planning screen layouts (opening windows, placing cursors, setting limits etc.) and is printed in XOR mode in order to preserve the screen image when it is unprinted.

GRID may be used without any parameters - in which case it will cover the whole screen area with gridlines every 32 pixels/lines. If however one parameter is given, that parameter is interpreted as the grid size (minimum 8). When two parameters (window and gridsize) are specified, the grid will displayed only within that window (the second parameter is needed when specifying a window for the grid).

HEX$ (no_of_hex_digits, value)   (Function)
DEC ("hex$")   (Function)

HEX$ returns the hexadecimal equivalent of a positive denary integer. The number of hex digits required must be stated (between 1 and 8).
Example: PRINT HEX$ (8, 179)   . . . gives 000000B3

DEC returns the denary (decimal) equivalent of a hexadecimal string. The hex must be enclosed by inverted commas;
Example: PRINT DEC ("20000")   . . . gives 131072

DUMP   (Procedure)

This is an extremely slow screen dump for use with Epson compatibles. DUMP will operate in mode 4 or 8 and will dump the whole screen to a printer connected to ser1_ (serial port).

Note: The process may be interrupted during the course of operation by holding down the ESCape key.

CNUM (colourA, colourB, stipple) (Function)

CNUM followed by two colour values (0 to 7) and a stipple pattern (0 to 3) will return the equivalent colour/stipple combination as a single value (between 0 and 255).

GETBRUSH x, y (Procedure)
PUTBRUSH x, y (Procedure)

These two procedures are for "reading from" and "writing to" the screen according to a definable brush. The preset brush shape is circular with centre at co-ordinates x,y. Brushes may be redefined by directly POKEing new values (see later).

GETBRUSH reads the screen pixel by pixel along a set course covering the area beneath the brush. The pixel colours are read into a buffer to be used by PUTBRUSH. PUTBRUSH places the pixels (read by GETBRUSH) onto the screen at specified co-ordinates x and y. This process is used by the program "transfer_x" to copy free hand from one position to another on the screen.

Example: 100   LIMXY 8, 8, 504, 228 : SETXY 240, 120 : XYSTEP 4, 4
            110   WINDOW 512, 256, 0, 0 : OVER -1
            120   REPeat loop
            130   MOVXY
            140   IF SPA : GETBRUSH XP, YP : PUTBRUSH XP+100, YP+20
            150   FOR c = 1 TO 2 : CROSS XP, YP, 7 : END FOR c
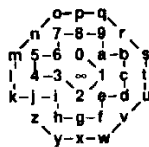   160   END REPeat loop

---

BRUSHBUF (Function)

BRUSHBUF gives the start address of the buffer used to define the path between each constituent point of the brush. The format of the buffer is as follows :

	Word	-	number of steps to take within the brush
then ...	Byte	-	parameter for first x (horizontal) step
	Byte	-	parameter for first y (vertical) step . . .and so on for each step . . .

A negative movement in x or y (ie: left or up) must be indicated by setting in addition the 8th bit of that byte (this is done by adding 128 to the number to move). In the default brush the path starts at the centre and spirals outwards, the first point read being the pixel just above the centre, and the last being that of the centre. The buffer starts with a word value 37, then bytes 0, 129 (right 0, up 1) for step 1. Then 1, 1 (right 1, down 1), and so on. The last step will be 2, 130 (right 2 pixels, up 2) to go from the outside (z) to read the centre pixel.

(The max. number of steps definable is 70, and the max. distance for each step is 127).

BCOLBUF (Function)

BCOLBUF returns the address of the buffer used for storing the colours read by GETBRUSH.
Each pixel colour is stored as a word in sequence following the path taken by the brush (see BRUSHBUF).

SVAR n, value (Procedure)
VAR (n) (Function)

SVAR and VAR provide a means of stocking a 32 bit value in memory which will not be erased by the commands CLEAR or NEW, or by loading another Basic program. They may be used for transfer of variables between programs or into compiled programs, storing ALCHP addresses or any other variable function. 16 variables are available.

To set the value of one of these variables, type SVAR - followed by the identifying number (from 1 to 16) and then the value you wish to store.

Example: Type -   SVAR 5, 131072   . . . then type NEW
           Type -   PRINT VAR(5)   . . . which will give a value 131072

FLEN (#n) (Function)

Providing a channel is opened to a directory device file, FLEN will return the length in bytes of that file. It is a useful check before loading code into memory so that the exact amount of space can be reserved.

The procedure is as follows:

Open the channel   - eg: OPEN#3, mdv1_program
Call the function   -   length = FLEN (#3)
Close the channel   -   CLOSE#3

WAITMD (Procedure)

The command WAITMD will suspend a program while any of the microdrives are turning. This can be used to avoid situations where screen messages appear before intended because the microdrive motors are still running.

Example: 100   SAVE mdv2_program
            110   PRINT "Wait for the motor to stop . . ."
            120   WAITMD : PRINT "Now the motor has definitely stopped"\
                  "You can remove the cartridge from mdv2_"

page 13

NOKEY

NOKEY will suspend any action while any of the keys are pressed. It can be used to prevent multiple exits from nested loops etc. when polling the keyboard using the function KEYROW.

Example: 100   REPeat loop
           110   AT#0, 0, 0 : PRINT "Press 'SPACE' "
           120   IF KEYROW(1) : AT 0, 0 : PRINT "- - - - - - OK - - - - - -" : NOKEY
           130   END REPeat loop

STORE w, h, x, y, address
PLACE w, h, x, y, address, [n]
STORSIZ

(Procedure)
(Procedure)
(Function)

These routines are intended for storing part or all of the screen into memory, and placing from memory to a position on the screen. The co-ordinates for the rectangle (width, height, x, y) defining the image area, assume a 512x256 pixel display. STORE can be used before opening a window on screen, to store any images that lie beneath that window. When the window is no longer required, PLACE may be used to restore the image to the screen (see "pointer_demo_bas" which uses this principle).

Another use for STORE/PLACE is to save a section only of screen to disk or microdrive (eg: a block 152x100 requires only 3800 bytes - after having used STORE or PLACE, STORSIZ will return the size in bytes of the screen block).

Example: 100   addr = RESPR (5000)
           110   STORE 152, 100, 40, 25, addr
           120   SBYTES mdv2_screen_section, addr, STORSIZ

The section may later be loaded from microdrive into memory and positioned on screen as desired using PLACE. It is however necessary to keep note of the proportions of the saved section, as severe distortion will occur if these are changed.

Horizontal co-ordinate parameters are rounded down to the nearest word interval for speed in moving screen memory. It is imperative that the parameters defining PLACE fall within the screen display area otherwise the QL will crash.

An optional fifth parameter [n] may be used with PLACE. It must comprise a value of 0, 1 or -1. If this parameter is zero, the section is moved to the screen normally regardless of the underlying image. If the parameter is 1 the two images will be 'OR'ed. If it is -1, the two images will be 'XOR'ed . . . which has a similar effect to printing with "OVER -1".

(Using a value of -1 can produce interesting results in mixing two or more images).

STORE, PLACE contd.

Another useful application is in panning or scrolling with "wrap-around" of opposite edges.

```
Example:   100   addr = ALCHP(80) : SETWIN 6 : CLS
           110   OPEN#3,scr_164x100a62x50 : BORDER#3,1,2 : INK#3,2 : LIST#3
           120   OPEN#4,scr_164x100a254x50 : BORDER#4,1,2 : INK#4,4 : LIST#4
           130   REPeat loop
           140   STORE 160,1,64,51,addr : STORE 160,1,256,148,addr+40
           150   SCROLL#3, -1 : SCROLL#4, 1
           160   PLACE 160,1,64,148,addr+40 : PLACE 160,1,256,51,addr
           170   END REPeat loop
```

COMP [address]   (Procedure)
DCOMP address   (Procedure)
COMPSIZ   (Function)

COMP is a screen compress routine designed to economise on the normal 32K required for storage. The savings on memory can be considerable, depending on the complexity of the image. However, very complex screens could require more than 32K. By using COMP without any parameters, the image will be processed but not stored.

The function COMPSIZ returns the number of bytes required to store the compressed screen. If this is satisfactory, COMP - followed by the address of a reserved area in memory, will store the compressed screen to that address.
The image can then be saved to microdrive if required by using (say) -

                              SBYTES mdv1_image_cmp,address,COMPSIZ

The procedure DCOMP together with a valid address will have the reverse effect, - the image is decompressed from memory to the screen. COMPSIZ will return the size of compressed image after either COMP or DCOMP.

```
Example:   100   REMark addr = ALCHP(20000)
           110   COMP
           120   IF COMPSIZ < 20000 : COMP addr : ELSE : PRINT "Screen too
                 complex ...  ";COMPSIZ;" bytes required"
```

PCOL (x, y)   (Function)

PCOL returns the colour of the pixel located at co-ordinates x,y. The co-ordinates must be relative to the whole screen area (ie: 512x256 pixels). In mode 8 the x parameter will be rounded down to an even value.
(Note - in mode 4, the colour returned will be 0, 2, 4 or 6).

For drawing routines, this function is extremely useful for pen colour changes etc. as colour can be indicated directly on screen by sample rather than by palette or menu selection.

MAG [#n], source_x, source_y, dest_x, dest_y   (Procedure)

Magnifies a section of screen by a factor of 4 in x and in y (the area is magnified 16
times). The area that is to be magnified measures 28x20 pixels - and is magnified to
112x80 pixels. The magnification is displayed within a window [#n] - if no window is
specified the default is #1. The parameters source_x, source_y are the co-ordinates of
the top left corner of the area to be magnified (relative to the whole 512x256 screen).
The next two parameters locate the top left corner of the destination block (relative to
the display window's origin - ie: 0,0 is the top left of the window #n). No error will take
place if the destination co-ordinates are wholly or partially outside the window's limits.

Example: 100   REMark   To magnify from the top right hand corner of the screen
             110   REMark   and display the magnification in window #0
             120   MAG#0, 484,0,0,0

Note: See the routine "zoom_bas" for an application example.

ROTATE   address, nbytes   (Procedure)

ROTATE will swap and rotate the bytes of the the first and last words of a sequence of
bytes (starting at the given address and ending at that address + nbytes), . . . then the
second and the second to last words of the sequence, then the third and third from last
and so on.

The principal use is in rotating a string of words of screen memory.
For instance: ROTATE 131072,32768 . . . will result in the entire screen being
effectively turned through 180°. ROTATE 133248,128 will rotate just one line starting
at address 133248 - which is the same as mirroring that line horizontally.

Example: 100   FOR y = 0 TO 255
            110   ROTATE 131072 + (128 * y), 128
            120   END FOR y

The effect of this will be to mirror the whole screen left to right. Using combinations of
individual line, and whole/part screen rotations, images can be turned or mirrored very
simply. It is important that nbytes (the length of sequence to rotate) is divisable by 4,
that is to say - an even number of screen words and that the start address is an even
number.

SFLASH n   (Procedure)

The parameter n is either 0 or 1 and has the effect of setting or resetting (zero-ing) all
the flash bits in the screen memory. In mode 8 - SFLASH 0 will cause any flashing to
cease. The main use of SFLASH is for reworking in mode 8, screen images created in
mode 4, to suppress flashing characteristic of the incompatible colour interpretation.

SHRINK address

SHRINK will store a reduced image of the entire screen at the specified address. The reduced "screen" will require 1/4 of the normal 32 K (ie: 8192 bytes) for storage (reserved by ALCHP or RESPR). This image can then be positioned on screen using the procedure PLACE. The new screen image will measure 256x128 pixels.

The procedure is not really suitable for use in mode 8 as flashing will probably occur. (However - this may be removed using SFLASH).

Example: 100 REMark   * * * * first load an image onto the screen * * * *
          110 addr = ALCHP(8192)
          120 FOR c = 1 TO 5
          130 SHRINK addr : DISP 0
          140 FOR x = 0,256
          150 FOR y = 0,128 : PLACE 256,128, x, y, addr
          160 END FOR x
          170 DISP SMODE/4 : PAUSE 100
          180 END FOR c

THE PROGRAMS

Transfer_x

This program is used for transfer and treatment of images between two screens. One unique feature is the ability to copy freehand from one image to the other, using a variety of brushes.

To run the program - type **EXEC_W mdv1_transfer_x**   (Note: as for all the programs on this cartridge, the toolkit extensions must be installed previously)

Transfer_x will not affect the current screen image - so it may be used from within a Basic program (memory permitting). Transfer_x reserves for itself around 42000 bytes to hold the second screen and for windowing etc.

Note: This will be released after exit from the program, providing no new directory devices are accessed during the program (due to common heap fragmentation - see ALCHP on page 4). This problem can be avoided by previously accessing all the devices likely to be used for SAVE-ing or LOAD-ing screens.

Transfer_x has two modes. When the program starts, you are in INLAY mode. The "help" window (which can be called during this mode only by pressing F1) is displayed. INLAY mode concerns functions such as loading and saving, positioning, panning, scrolling etc. of the two screens - ie: the main display, and the second screen (displayed behind a window inlayed in the main display).

Press a key to remove the help window.

Loading an image

Press F2 to load an image from disk or cartridge (either to the inlay or main screen). After indicating the device from which to load, a directory of files is shown. Enter the file name exactly as it appears in the directory (or just press ENTER if you do not wish to load a file). You will then be asked whether to load to screen or inlay.

Note: If you load a "mode 4" screen while using the program in mode 8 . . . severe flashing will occur. This may be removed by pressing the key "4".

Saving an image

After processing, the main screen may be saved to disk or microdrive. Press F3 and input the name of the device to save to. A directory of files already saved is displayed and you are asked for a name for the image to save. If there is not enough room to save the screen (64 sectors) or you have changed you mind - just press ENTER. Otherwise, type a name for the screen and press ENTER.

Note: SAVE-ing will overwrite any previously saved file of the same name.

Transfer_x contd.

Once both images have been loaded, the inlay window can be manipulated as follows:

←→ ↑↓ . . . . to pan/scroll the inlay screen
←→ ↑↓ + SHIFT . . . . to adjust the size/shape of the window
←→ ↑↓ + ALT . . . . to move the inlay window around the main screen
F5 . . . . will print the image in the window to the screen beneath

Note: The inlay window may be removed from the screen by reducing its height to
         0 using SHIFT and ↑ , after which ←→ ↑↓ may be used to pan/scroll the main
         screen display.

COPY mode
This mode is concerned with freehand copying, drawing, transfer and stretching of
screen images/textures. Press F4 to enter COPY mode.

You are presented with two superimposed cross-wire cursors which may be moved
around the display in parallel (using the CURSOR keys). One of these represents the
"reading" brush, and the other the "writing" brush. To set the relative positions of the
cursors, the function keys F1 to F5 are used as follows:

F1 - writing brush right
F2 - writing brush left
F3 - writing brush down
F4 - writing brush up
F5 - the writing brush assumes the same location as the reading brush

While the cursors are separated, pressing SPACE will transfer from one cursor to the
other, similar to a pantograph.

However - when the cursors are superimposed, the ALT key will cause the brush to
read the texture/colours into the buffer. Consequently, pressing the SPACE bar will
write those colours back to the screen. It is then possible to draw with the stored
texture.

Using this technique (and a skilful manipulation of ALT and SPACE), textures can be
stretched from (say) the side of a face to distort its shape/contours. Each time ALT is
pressed - a new set of colours is read by the brush into the buffer.

Step
For all cursor movement, the step rate may be altered by pressing SHIFT. Between 1
and 8 pixels per step are possible. While holding down SHIFT, the current step is
indicated at the top left of the screen.

Transfer_x   contd.

Brushes
A selection of brushes is available - to change brush press CTRL. While the CTRL key
is pressed the brush shape/icon is displayed at the top of the screen. Eight brushes
and two airbrushes are provided - which, when used in conjunction with varying step
sizes, gives an enormous range of possible textures for drawing, transfer etc.

Note: when using the airbrushes, a step of 2 or 3 is recommended.

To EXIT from COPY mode - press ESCape
To QUIT the program (from INLAY mode), and return to Basic - press CTRL + ESCape

Zoom_bas

Zoom_bas is a routine written in Basic to illustrate the procedure MAG, used to
examine a 28x20 pixel block magnified to 16 times. The routine (which can be
incorporated within a larger program) is used as follows:

A block appears on screen, which will move in response to the cursor keys. This
should be positioned over the area to be examined.
Press SPACE - the block will be magnified in a window to one or other side of the
screen.

Inside this window a flashing "pixel" indicates the current cursor position. It is possible
to draw within the window using the CURSOR keys and SPACE (the default ink colour
is black but may be changed by pressing "C"). Any drawing will be echoed normal size
on the screen. Moving the cursor to an edge of the window will cause the area that is
magnified in the window to be scrolled or panned in that direction. (Pressing F5
indicates the screen location of the block).

To exit from the routine press ESCape.

Definer_bas

This is a very simple character definer which (unusually) allows you to work with the
complete 9x8 character matrix (although only mode 4's CSIZE 1,0 and 1,1 are
capable of using the full matrix).

Each screen channel has two character fonts. Normally the first contains characters 31
(the graphic square) to 127 (©) while the second contains characters 127 to 191 (© to
↓). Font 1 requires 875 bytes and font 2 requires 587 (see FONT on page 8). When
the program is loaded you are asked to select font 1 or 2 (which is then transfered from
ROM into an area pointed to by VAR(3)). On loading a font from microdrive for
redefining, this selection is made automatically.

Definer_bas   contd.

The main display consists of four windows. In the top window the new set of characters is displayed beneath their ROM set equivalents. These may be scrolled left or right using F1 or F2. The current character is also displayed in the next window down in a selection of sizes together with its character code. To the left is an enlarged display of the character's pixels. On releasing F1 or F2, after a short pause - the current character will be read into the large display. (To prevent this - press TAB immediately after releasing F1 or F2).

You can draw within this display to modify the character, using the CURSOR keys and SPACE bar (F5 clears the matrix). You can write the modifications to the character definition by pressing ALT.

The fourth window contains "help" information, and is used for prompts when loading (F3) or saving (F4) character sets.

Note: Saving will overwrite a previously saved file of the same name.

Once a font has been edited or created and saved to disk or microdrive, it can be installed using the command FONT. The process is as follows:

        Reserve enough space - ie: addr = RESPR(875)
        Load the font   LBYTES mdv1_symbol_set, addr
        Install the font for (say) #2   FONT#2, addr, 0

Note: If either font address parameter is zero, that font will default to the ROM character set.

EXTENSIONS (F or P indicate either function or procedure)

Programs

Also on the GRAPHIC TOOLKIT microdrive . . .

Pointer_demo_bas   Puzzle_bas
RPRINT_bas   Relief_scr
Symbol_scr   Screen_scr