FPCS copy

VORTEX by Chris Blackmore.

WHAT VORTEX IS FOR:

Vortex is a set of machine code graphics routines written for use in conjunction with Nascom ROM BASIC, its purpose being to speed up the display of pixel graphics. Two different "pictures" may be held in store by Vortex, and manipulated in a variety of ways, thus enabling the addition of animated graphics to programs. Routines to save and restore any text on the screen are also provided, so that mixed text and graphics displays can be produced.

SYSTEM REQUIREMENTS:

Vortex will run on any Nascom system that complies with the following requirements:

1/ There must be sufficient RAM. Versions of Vortex for 16K, 32K and 48K Nascoms are available.

2/ Nascom ROM BASIC must be fitted. If the modifications listed in a later section are carried out, Nascom Tape BASIC may be used instead.

3/ The character generator must be capable of producing the standard Nascom 2 pixel graphics character set. For example, a Nascom 1 fitted with a Bits & PC's graphics unit (programmed to produce the character set shown in a later section) could be used.

4/ Since Vortex does not use any monitor routines, any monitor except Nasbug T2 may be used. (T2 does not work with Nascom ROM BASIC)

GENERAL NOTES:

Vortex gives a screen resolution of 96 pixels horizontally by 48 pixels vertically. X coordinates start at Ø at the left of the screen, and rise to a maximum of 95 at the right. Y coordinates start at Ø at the bottom of the screen, (unlike the Nascom BASIC system, which starts on the line next to the top) and rise to a maximum of 47 at the top. Vortex does not "crash" when supplied with coordinates outside the range just stated. Any point which would be off the screen is ignored, but very large values, such as 250, may produce unexpected images if they occur. It is recommended that programs include code to prevent such values from being passed to Vortex.

In use, Vortex occupies the top 4K of the system RAM. The first 1½K of this space is the Vortex code; the remainder is used for the two "screens" of graphics data in a compressed format, the text store, and the Vortex scratchpad and stack. There is also some spare memory, which may be used in any manner desired; its addresses are given in a later section.

PREPARING TO USE VORTEX:

First, "cold start" the BASIC, setting the memory size as follows:

For 16K systems...........16384
For 32K systems...........32768

For 48K systems...........49152

Then re-enter the monitor, and load Vortex into the space you have just reserved at the top of the memory. Return to BASIC with a "warm start", and enable Vortex. This is done by DOKEing the address of its first byte into the location called USRLOC. The command required is as follows:

For 16K systems.......DOKE 4100,16384
For 32K systems.......DOKE 4100,-32768
For 48K systems.......DOKE 4100,-16384

If Tape BASIC is being used, the 4100 is changed to 12292 in each case. Vortex may be enabled in immediate mode, or as part of a program. Once it has been enabled, it will remain so unless action is taken to turn it off, by changing the contents of USRLOC. It is possible to use Vortex with no further preparation, but considerable savings in program length and run-time can be made by storing the addresses to which coordinates etc are passed as variables. See the example programs for how to do this.

DESCRIPTION OF VORTEX ROUTINES:

The routines of which Vortex is composed have numbers, by means of which they are called. For example, if it is desired to call routine number 5, include the code U=USR(5) in your program. BASIC will then call the first routine in Vortex, which recovers the argument (ie the 5 in the example given) and uses it to locate and call the routine required. Routine numbers greater than 28 will be ignored, unless extra routines have been added by following the procedure given later.

It is important to remember that only the routines numbered Ø, 27 and 28 act directly on the screen; all the others act on the storage space used by Vortex. Because of this, quite complex processes can be made to happen instantaneously, as their results only become apparent when the screen is updated.

USR(Ø) <u>UPDATE</u>

This routine updates the screen, using the compressed graphics information stored in the main Vortex RAM block (the first one, in fact). If a 1 has previously been POKE'd into the location called TOPLIN, the top line of the screen will be updated; if TOPLIN contains a Ø the top line will not be disturbed. If there is text on the rest of the screen, which it is desired not to destroy, routines 27 and 28 should be used to save and restore it before and after the update.

USR(1) USR(2) USR(3) <u>SETXY RESXY FLPXY</u>

These routines set, reset or flip (ie, change black to white, or white to black) a single point in the main RAM block. Before they are called, the coordinates of the point must be POKE'd into the locations X1 and Y1.

USR(4) <u>TESTXY</u>

This routine is called when it is necessary to find out whether the point whose coordinates are in X1 and Y1 is set or reset. The result is passed back to the BASIC program; if for example the routine was called by using X9=USR(4) , then the value of the variable X9 will become 1 if the point was set, Ø if the point was reset, or -1 if the coordinates referred to a point not on the screen.

USR(5) USR(6) USR(7) SETLIN RESLIN FLPLIN

These routines set, reset or flip the line joining the two points whose coordinates have been put in X1,Y1 and X2,Y2; an algorithm which gives a close approximation to a straight line is used.

USR(8) USR(9) USR(10) SBLOCK RBLOCK FBLOCK

Here, a rectangular block of points is set, reset or flipped. The coordinates of the bottom left hand corner must first be put in X1 and Y1, and those of the top right hand corner in X2 and Y2. If X1 is greater than X2, or if Y1 is greater than Y2, then nothing is done by the routine.

USR(11) USR(12) USR(13) SETALL RESALL FLPALL

These routines are used to set, reset or flip all the points in the main Vortex RAM block. RESALL is used to clear the main block, prior to use.

USR(14) BSWAP

When this routine is called, the contents of the main RAM block and the secondary block are exchanged. This routine is particularly useful for producing the appearance of motion in the display.

USR(15) BCOPY

This routine copies the contents of the main RAM block into the second block.

USR(16) USR(17) USR(18) BAND BOR BXOR

These routines are used to combine the two blocks in logical "AND", "OR" or "EXCLUSIVE-OR" modes. The result of the operation is stored in the main RAM block.

USR(19) USR(20) USR(21) USR(22) SLIDER SLIDEL SLIDED SLIDEL

When one of these routines is called, the contents of the main RAM block are moved one pixel space to the right or left, or up or down. If the location called WRAP has been POKE'd with a 1, the row of information which leaves the edge of the screen woll reappear at the opposite edge. If WRAP contains a Ø, however, the information will be lost.

USR(23) GROW

When this routine is called, the central part of the main RAM block is doubled in size in one of the following ways. If the location called GTYPE has been POKE'd with a 1, growth will be in a horizontal direction only; if it has been POKE'd with a 2, growth will be vertical only; setting GTYPE to 3 causes both horizontal and vertical growth to be carried out.

USR(24) SHRINK

This routine is the reverse of GROW. The location called STYPE is used to control its effect in the same way that GTYPE controls the operation of the GROW routine.

USR(25) REVERS

When REVERS is called, the information in the main RAM block is reversed from left to right.

USR(26) INVERT

This routine has the effect of inverting the contents of the main RAM block.

USR(27) USR(28) SAVTEX RESTEX

When SAVTEX is called, all non-graphic characters on the screen are copied to an area of memory immediately above the two RAM blocks. Calling RESTEX, which would usually be done after a screen update, would restore the text to the screen.

MAKING MODIFICATIONS TO VORTEX:

It is possible to add the addresses of your own machine code routines to the address table within Vortex. The first free byte of this table is the 94th byte of the Vortex code, and addresses are stored low byte first. There is room for the addition of twenty-one addresses. Since Vortex checks the routine number, you will also have to amend the section of the program that does this; increment the twentieth byte of the code once for each address you add.

In order to use Vortex with Tape BASIC, amend the 14th byte of Vortex from E9 to 19, and the 396th byte from F0 to 20.

It is not possible to transfer Vortex to EPROM without rewriting parts of it. This is because the parts referred to are self modifying, to save space. (The author is well aware that this is considered to be bad practice, but is also aware of how much memory has been saved.)

It is unwise to attempt to relocate Vortex, as some of the routines have been made to run faster by writing them in such a way that they only work when they are in the right places.

The table from which the character codes are obtained starts at the 268th byte of Vortex, and should only need to be amended if you have a totally non-standard graphics system.

CHARACTER SET REQUIRED BY VORTEX:

In this illustration, the shaded parts are those NOT lit up on the screen.

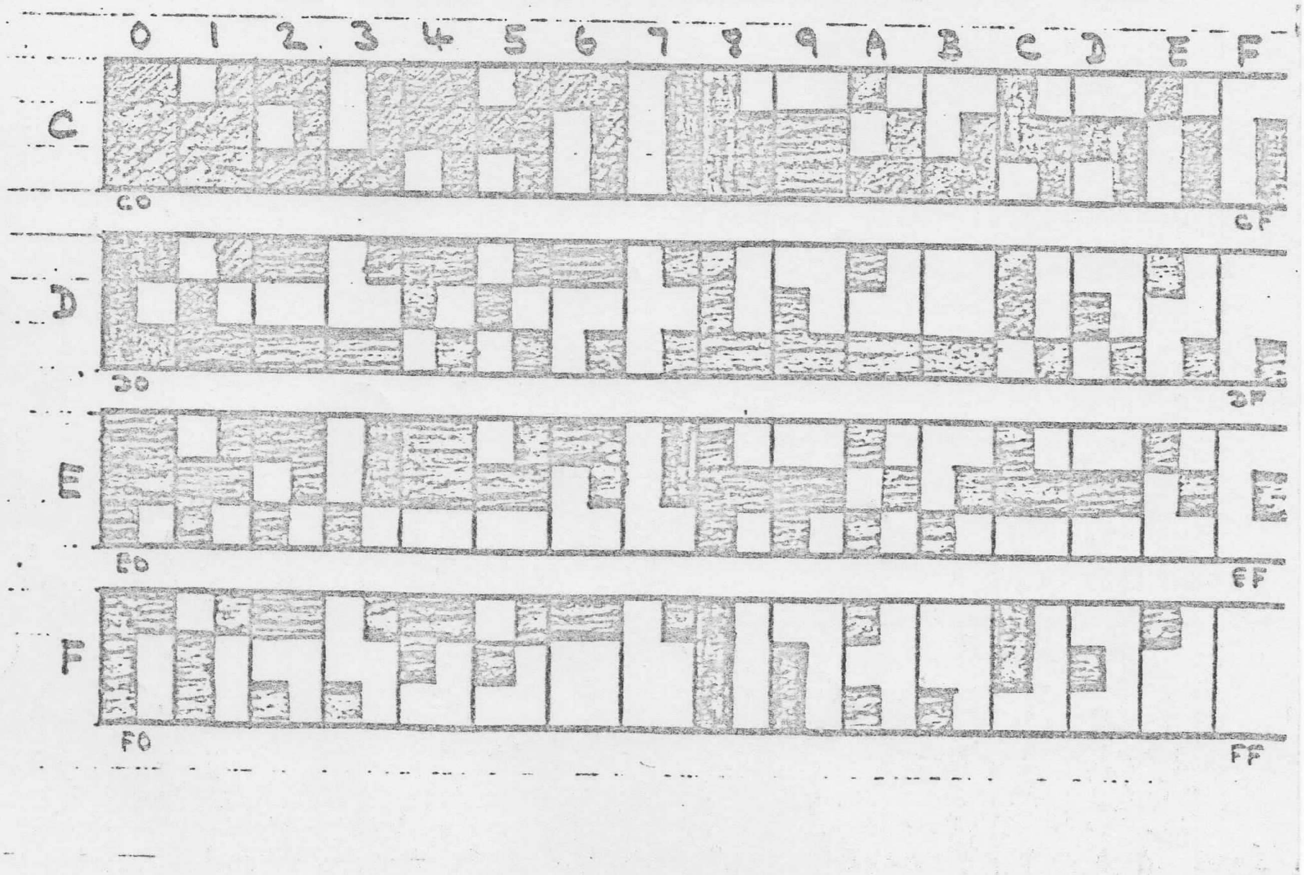

TABLE OF SYSTEM ADDRESSES FOR ALL VERSIONS:

LOCATION	16K VERSION		32K VERSION		48K VERSION	
	HEX	DEC	HEX	DEC	HEX	DEC
X1	457A	17786	857A	-31366	C57A	-14982
Y1	4579	17785	8579	-31367	C579	-14983
X2	457C	17788	857C	-31364	C57C	-14980
Y2	457B	17787	857B	-31365	C57B	-14981
TOPLIN	457D	17789	857D	-31363	C57D	-14979
WRAP	457E	17790	857E	-31362	C57E	-14978
GTYPE	457F	17791	857F	-31361	C57F	-14977
STYPE	4580	17792	8580	-31360	C580	-14976
Main block						
From	4590		8590		C590	
To	47CF		87CF		C7CF	
Second block						
From	47D0		87D0		C7D0	
To	4A0F		8A0F		CA0F	
Text store						
From	4A10		8A10		CA10	
To	4D0F		8D0F		CD0F	
Spare memory						
From	4D10	19728	8D10	-29424	CD10	-13040
To	4FBF	20415	8FBF	-28737	CFBF	-12353
Vortex stack						
From	4FC0		8FC0		CFC0	
To	4FFF		8FFF		CFFF	

See the example programs, where use is made of the decimal addresses shown above. The addresses below are of use if the Vortex code is being modified as discussed on page 4.

First free byte of routine address table:			
	405D	805D	C05D
Number of addresses in table:			
	4013	8013	C013
Addresses to change for use with Tape BASIC:			
To 19	400D	800D	C00D
To 20	418C	818C	C18C

EXAMPLE PROGRAMS:

The first example program puts a rectangular block on the screen, and then makes it grow and shrink alternately.

```
100  CLS
105  REM First set variables pointing into Vortex scratchpad.
110  REM Values shown are for 32K systems- amend if necessary.
115  X1=-31366: Y1=-31367: X2=-31364: Y2=-31365
120  TL=-31363: WR=-31362: GT=-31361: ST=-31360
125  REM Now enable Vortex- 32K version again!
130  DOKE 4100,-32768
140  REM Reset both blocks of RAM using RESALL and BCOPY.
150  U=USR(12): U=USR(15)
160  REM Set the coordinates of two points.
170  POKE X1,Ø: POKE Y1,Ø
180  POKE X2,90: POKE Y2,45
190  U=USR(8): REM This sets a rectangle in the main block.
200  POKE TL,1: REM So that topline will be updated.
210  POKE ST,3: POKE GT,3: REM Select GROW and SHRINK types.
215  REM The action starts here!
220  FOR I=1 TO 4
230  U=USR(24): U=USR(Ø)
240  GOSUB 1000
250  NEXT
260  FOR I=1 TO 4
270  U=USR(23): U=USR(Ø)
280  GOSUB 1000
290  NEXT
300  GOTO 220
1000 FOR J=1 TO 100: NEXT: RETURN
```

(Line 1000 is a delay routine; not a common sight in BASIC programs with graphi
The program can be made less boring by amending line 190 to read:

```
190  U=USR(5)
```

and adding the following line:

```
255  U=USR(25)
```

EXAMPLE PROGRAMS.

The second example is the author's favourite attempt at computer "art", and consists of most of the routines in Vortex called at random, with certain restrictions. To use it with systems other than 32K, lines 110, 120 and 130 must be amended, using the values given on page 5.

```
100 CLS
110 X1=-31366: Y1=-31367: X2=-31364: Y2=-31365
120 TL=-31363: WR=-31362: GT=-31361: ST=-31360
130 DOKE 4100,-32768
140 POKE TL,1: POKE WR,1
160 DEF FNR(Z)=INT(RND(1)*Z+1)
170 POKE X1,FNR(95)
180 POKE Y1,FNR(47)
190 POKE X2,FNR(95)
200 POKE Y2,FNR(47)
210 V=FNR(28)
211 IF V=11 OR V=12 THEN 210
212 U=USR(V): U=USR(Ø)
230 IF V=19 OR V=20 OR V=21 OR V=22 THEN 250
240 GOTO 170
250 FOR I=1 TO FNR(10)
260 U=USR(V): U=USR(Ø)
270 NEXT
280 POKE GT,FNR(3): POKE ST,FNR(3)
290 GOTO 170
```