TI*MES

SPRING: 1999

ISSUE:64

Celebrating 20 years of TI-99/4 & TI-99/4A

TIUG
AGM '99
10/4/99

Committee Members

Position	Name	Address	Telephone
Chairman and Sysop of group BBS	Trevor Stevens	249 Southwell Road East, Rainworth, Notts. NG21 0BN	☎01623 406133 FAX:01623 406134 BBS🖳01623 406135
Vice Chairman	Mark Wills	41 Broxtons wood, Westbury , Near Shrewsbury, Shropshire. SY5 9QR	☎01743 885049
General Secretary & TI*MES Editor	Richard Twyning	24 Peel Road, Mansfield, Notts. NG19 6HB	☎/🖹FAX 01623 453934 ☎Mobile: 0467 445658 🖹FAX:0467 449009
Interim Newsletter production, web page design, disk library custodian.	Richard Speed	213 Comptons Lane, Horsham, West Sussex. RH13 6BZ	☎01403 242853
Also web Page Design	Ian Pare	10 Sotheby Avenue, Sutton-In-Ashfield, Notts. NG17 5JX	☎01623 552549 🖹FAX: 01623 452729
Treasurer and group membership	Alan Rutherford	13 The Circuit, Wilmslow, Cheshire. SK9 6DA	☎01625 524642
Module and Cassette Librarian	Francesco L. Lama	14 Granville Court, Cheney Lane, Oxford. OX3 0HE	☎01865 721 582
Hardware and TI*MES publication	Ross & Christine Bennett	20 Oak Avenue, Romiley, Stockport. SK6 4DN	☎0161 4307298 🖹FAX: 0161 4834516

CONTENTS

Members on the Internet

Alan Bray: alan@	ɔn.co.uk
Francesco L. Lama:	ebl.rl.ac.uk
Ian Pare: ian@ir	ı.co.uk
Alan Rutherford: abr@	ɔnline.co.uk
Gary Smith: gary.s	h.co.uk
Mark Wills: markwill	ɛrve.com
Richard Speed: rich	sql.com
Stephen Shaw: stephe	ternet.com
Trevor Stevens: steve	tmail.com
Richard Twyning: 007@ı	ɲon.co.uk

Rambles. Stephen Shaw. Nov 1998

Greetings.

This is too late to be current, but I have to record my appreciation for the hard work done by a small number of Group members in organising the TiRef in Nottingham. Meetings of this sort do not just happen, there is an enormous amount of work involved. So, thanks!

It has been a while since I was able to attend a TI gathering, but with the death of my father in June, I am now able to get out more, and made the journey to the TiRef.

It was fun. I purchased a copy of PC99 from Mike Wright, and got to meet people I have never before met- such as Malcolm Adams, the author of so many excellent TI programs. After 15 years I get to meet him!

With so little usage of the disk library of late, it is now "closed down" as the TI disks have been relocated for the purpose of placing on CD in one format or another. Keep watching TI*MES. It is going to be a major job, so there may be some delays...

Meanwhile I now have the full Tigercub Public Domain collection on a 100 meg Zip Disk in PC99 format, and will be posting samples from this collection on my web site, so pay a visit from time to time!

http://www.btinternet.com/~shawweb/stephen/TI.htm and that is capital T and capital I at the end! All other characters are lower case.

INFOCOM

Some readers will have purchased and played the Infocom Text Adventures on their TI's. I was lucky enough to spot an Activision CD Rom in my local software shop with some 33 Infocom adventures, and under ten pounds.

I cannot remember how much my TI Adventures cost but I am sure it was well over 33p per game!

If you see the Activision Infocom package, ignore the system requirements on the box- it runs in DOS from the CD, and requires NO hard disk usage at all. Minimal need for dos memory. However you do need the Acrobat reader to get at the documentation in .PDF format - if you dont have it the CD has versions for W9,W3 and DOS.

The .pdf files take up 1050 pages if you print them all out...

The Infocom adventures run very nicely in DOS6, and even NT4 is happy with them but for W9 you will need to run them in dos mode, as running in a normal dos box seems to disconnect the keyboard!

Note that the collection CD does NOT contain the Hitchhikers Guide to the Galaxy, but we have that on the TI anyway and can port it to PC99.

I became very used to the TI format of joystick, and have not really been happy with any PC joystick I have seen, which offer far too great sensitivity and far too little tactile feedback. I was pleased to find a "kids" joystick from Saitek, the X1-30, which offers 8 positions and two fire buttons. This reacts much more like the traditional arcade machine joystick and I now have one connected. Very inexpensive too.

PC-99

Now I have PC99 I find it easier to use the TI on a PC than on a TI, the emulation is good, and it is easier to alter hardware configurations.

Of course, emulating software is one thing, emulating hardware is another, and with such very different hardware configurations there are difficulties, which PC99 handles with great aplomb.

Sprites- a PC technically has one sprite, on the TI the VDP chip handled up to 32 sprites whilst the CPU carried on with the program. So we have to emulate TWO processors on one... not so easy. Back with Extended Basic initial release, TI set all sprites active even if you werent using them- this kept sprite control perfectly in line with processor control. With Version 110 of ExBas, which most people use, the number of active sprites depended on how many you were using, which caused sprite motion to become disconnected from cpu speed. It was not found to be noticeable on the TI.

On a PC emulation, where one cpu does everything, running extra sprites will cause a change in relative speeds, especially in XB programs. PC99 gives you the tool to handle that- you can (on the fly as it were) amend the relationship between sprite speed and cpu speed very easily to obtain obedient sprites.

This also fixes a difficulty we had on the TI with XB programs from the USA comuing to the UK. There was a different relationship between cpu speed and sprite speed between USA and UK consoles. You had to go through each program and rewrite sensitive sprite speeds. What a pain!

Even then there was some variation between UK consoles! PC99 allows you to fine tune sprites in a way you never could on a real TI.

The TI used a matrix keyboard and no buffer whereas the PC uses a serial keyboard and a buffer. Many TI programs used and relied upon the "split keyboard" mode. Now the emulation software handles the different CALL KEYs, but the PC keyboard only sends back one key press at a time to the emulator!

You can write programs for the PC to remap the keyboard but there are several different types of PC keyboard, and you lose full compatibility if you use direct addressing for the keyboard.

Which brings me to MICROPINBALL 2. Excellent program! Mike Wright has written an alternative keyboard routine to enable key Q to tell the emulator to raise both flippers. You need to reload the standard keyboard map for any other program though! Other programs with split keyboard response will have similar problems.

Micropinball 2 is also a very tightly written machine code program.

For most modules, on my 266Mhz PC I run the emulator at 40% speed (again this is very easily changed on the fly) but on MicroPinball I can go to 5% speed and not notice it being slow.

Also in order to prevent the pinball flipper form "kicking" I found I had to extend the keyboard delay up to a quite extravagant 57,000

(fifty seven thousand).

I think you have the idea- the defaults will generally run modules ok, but 9900 machine code programs will probably need slowing down a lot, whilst Basic programs, even running at 100% speed may still seem slow compared to a PC program!

PC99 makes it very easy to fine tune the emulator to get the best from a TI program, with rapid adjustments possible to program speed, keyboard response, and sprite speeds.

The TI had a sound chip which a PC lacks, and as most PCs have a sound card, the CALL SOUND commands have to be translated to sound card data, AND find out how to communicate with the soundcard (there are many possibilities on soundcards!).

I have a very modern Soudblaster 64 AWE plug-n-play soundcard. If I run PC99 under W98 in a dos window, the config program cannot find my sound card and defaults to silence if I try to set the soundard.

However running PC99 in Dos 6.22 I am able to tell it to use my Soundblaster sound card, and subsequently, in W98, it doues so without a worry.

There seems to be problems with TI programs where the program speed drives the sound - eg using negative sound values. For example I was unable to run the Puppytown Boogie in XB. And don't think of playing the Harrison music disks- even at 1% of program speed they still play much too fast! The German "pop music" disk runs quite well under PC99 however.

Which operating system to run PC99 in??? It seems to be happiest with Dos6. Running in a dos box in W9 there is some small lack of smoothness in operation (part of W98) but you can have several dos windows open at once whichallows maximum flexibility for disk access. Do not think of running PC99 under NT, they are essentially incompatible, as NT protects itself from aggressive DOS programs (in particular any trying to write directly to a sound card). My NT4 set up also had difficulty locking the correct video mode.

A TI Disk is a simple Dos file- for example TIDISK.DSK, and you merely tell the emulator that (for example) DSK1 is the dos file C:\PC99\DSK\DSKFILES.DSK or whatever. You can change the paths to DSK1, 2 and 3 on the fly using simple commands. And as a dos file they can be located on any storage medium- floppy, hard disk, zip disk, cd-rom. All very easy.

PC99 has some excellent utilities for use in DOS, which enable you to convert TI Emulate/V9T9 files/disks to PC99 format; look at the directory of your TI disk in a dos file; and search all your TI disks to find a particular program- very very useful when you have hundreds of TI disks and want to know where Wumpus is hiding!

The documentation is in various formats, including TXT (ascii) format, Wordperfect 5.1 (which most word processors can read) and Acrobat PDF files- warning: Mike uses Acrobat 3, and many of the A3 files cannot be read with the Acrobat Reader v2.1 which is given away so much here in the UK. You can sometimes find the Version 3 reader around- it is in any case entirely free. Version 3 allows more compact files than Vn 2.

I have no hesitation at all in recommending the PC99 emulator, provided your PC meets the specs. As indicated above I find on my 266Mhz PC I can run most modules at 40% speed, but even 100% can be slow on Basic programs!

The basic package has all you need to run Basic, Extended Basic and Assembler programs (DF80 or program format). You can pay a little extra for the modules/disks to be emulated to run TI Logo, uscd Pascal or TI Pilot

etc. You can emulate either the Myarc or TI ram card (or others) but note that the Myarc ram card is not compatible with the Plato module- so much easier to change ram cards in the emulation!

This is an incompatability in the original hardware not an emulation bug.

I have four versions emulated of Extended Basic (TI 110, Mechatronic, Super Extended Basic and Myarc 212) which gives me a lot of power.

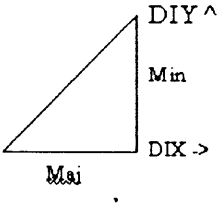
Be sure to ask for some of the manuals if you dont have them- available at low cost from CaDD in Version 3 .pdf acrobat format. Get a new Assembly Manual or TI Logo Manual!

CaDD also offers an inexpensive conversion service for your TI disks if you have data you cannot get from elsewhere and need to port over.

An exceptional product and service, deserving of ten stars out of ten.

The graphics modes of the V9938
part 2
By Alan Bray

Last time we looked at the 'LINE' command, but I didn't tell you how it really worked. So now we will go through the command explaining how the registers are used and also giving you your first taste of using the status registers. (It may be handy to have the list of control registers from the last article with you).
The line to be drawn is the hypotenuse of a triangle found by defining long and short sides from a single point.



Setting up the command

MXD: Select destination memory. 0=vram 1=expansion ram
Maj: No of dots in long side (0-1023)
Min: No of dots in short side (0-511)
MAJ: Direction for long side. 0=x-axis. 1=y-axis or the major side equals the minor side.
DIX: Direction from source to end point. 0=right. 1=left.
DIY: Direction from source to end point. 0=down. 1=up
DX: Basic x-coordinate (0-511)
DY: Basic y-coordinate (0-1023)
Then you must write >7 into the upper four bits of the command register (R46). In the lower four bits you can wite a 'logical' operation. (I'll deal with these logical operations in a future article).
After the 'LINE' command has been executed (e.g. LI R0, >2E7x, BLWP @VWTR (the 'x' is the logical operation)), Status register 2 (S#2) will have to be checked. The bit we are interested in is the CE bit, which will be set to 1 while the LINE command is being executed and reset to 0 when the command is finished. You can see how this is done by looking at the last 9 statements of the LINE command in the last Ti*mes.

Setting up the registers

R36 | DX7 | DX6 | DX5 | DX4 | DX3 | DX3 | DX1 | DX0 |
R37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | DX8 |
R38 | DY7 | DY6 | DY5 | DY4 | DY3 | DY2 | DY1 | DY0 |
R39 | 0 | 0 | 0 | 0 | 0 | 0 | DY9 | DY8 |

The above registers specify the x and y basic co-ordinates.

R40 | MJ7 | MJ6 | MJ5 | MJ4 | MJ3 | MJ2 | MJ1 | MJ0 |
R41 | 0 | 0 | 0 | 0 | 0 | 0 | MJ9 | MJ8 |

The above specify the NX (Maj) ...No of dots in the long side.

R42 | MI7 | MI6 | MI5 | MI4 | MI3 | MI2 | MI1 | MI0 |
R43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | MI8 |

These are the NY or No of dots in the short side.

R44 | 0 | 0 | 0 | 0 | C3 | C2 | C1 | C0 | Colour Register set-up for G4 and G6 modes.
| 0 | 0 | 0 | 0 | 0 | 0 | C1 | C0 | Colour register set-up for G5 mode
| C7 | C6 | C5 | C4 | C3 | C2 | C1 | C0 | Colour register for G7 mode
As well as being the colour register, R44 is also used as the place to put the first byte of data to be transferred from CPU to VRAM.

R45 | 0 | - | MXD | - | DIY | DIX | - | MAJ |

MXD = destination memory select. DIY= end point (y). DIX= end point (x). MAJ= long side diection select. The bits marked with ' - ' are not used in the LINE command.

R46 | 0 | 1 | 1 | 1 | LO3 | LO2 | LO1 | LO0 |

This is the command register. The most significant four bits contain the LINE command (>7), and the least significant four bits can contain a logical operation to be done on the colour.

Status Register 2

| TR | VR | HR |BD | 1 | 1 | E0 | CE | It is only the CE bit that we need to check in the LINE command . This is the ANDI Rx, >0100. I will have to devote a future article to the status registers as they will take up quite a lot of room in the magazine.

Writing to the registers

The registers can be written to directly by first sending the data and then the register number to Port1. The register number is in the least significant six bits. The most significant two bits are set to 10, so the value of this second byte would be 10xxxxxx , where xxxxxx contain the register number (0-46) . Port1 is also used to set the Vram address. The most significant bit of the second byte sent to the port is the address/register flag and determines the operation that is to take place. When the bit is set to '1' , writing data to a control register takes place. You can also use the indirect method to write the data either to the same register or to increment the register. For instance, to send data from the CPU to the VRAM you would need to use R44 to set the first byte of data in. Then you could send all the following bytes to R44 using the indirect method, or, you could write data consecutively to R32 - R40 to set up your command registers. So to write to the same register, store the register nmber in R17 (the control register pointer), set the most significant two bits to 10. So the binary value in R17 would be - 10xxxxxx - where 'x' is the register number you want to continually write to. When this is done you can send your data to this register by writing your data to port3 (examples of all these methods will eventually be covered).

If you want to auto-increment the registers written to, then set the most significant bits to 00xxxxxx, where 'x' is the beginning register number. This way you could write to control registers 0 - 23 in one go, so you could set up screen modes very quickley.

Accessing the Vram.

The first 64k of VRAM and the 64k of expansion ram both share the same address. Bank switching is used so that they can both be on line at the same time. Bank switching is controlled by bit 6 of R45, 0=VRAM, 1=XRAM.

The 17-bit address for the 128k of VRAM is set in the address counter (A16 to A0). R14 contains the high order three bits of the address (A16

to A14). So this register can be used to switch between eight 16k pages. The 14 low order bits should then be sent to port1 in two bytes. The first byte is A7 to A0, and the second byte is A13 to A8, but, the most significant two bits of this byte signify read or write (binary 00=read, 01=write), if there is a carry from A13, then the data in the register is auto-incremented. This auto-incrementing doesn't apply to G1, G2, MULTICOLOR< and TEXT1 modes.

The whole of the VRAM can be accessed as x-y co-ordinates, but only 212 lines at a time. A set of 212 lines is referred to as a 'page. The 'page' can be selected from R23 which sets the location of the first line to begin the display at. The Pattern Name Table will be set in R2.

G4 **address** **G5**

(0,0) (255,0) >00000 (0,0)
(0.511)
page 0
page 0
(0,255) (255,255) (0,255)
(511.255)

(0,256) (255,256) >08000
(0,256) (511,256)
page 1
page 1
(0,511) (255,511)
(0,511) (511,511)

(0,512) (255,512 >10000
(0,512) (511,512)
page 2
page 2
(0,767) (255,767)
(0,767) (511,767)

(0,768) (255,768) >18000
(0,768) (511,768)
page 3
page 3
(0,1023) (255,1023)

(0,1023) (511,1023)

>1FFFF

Using this 'page' concept G6 would be ; page 0 - (0,0) to (511,255), address >00000 to >FFFF

Page 1 - (0,256) to (511,511), address >10000 to 1FFFF

And G7 would be; page 0 - (0,0) to (255,255), address >00000 to >FFFF

Page 1 - (0,256) to (255,256), address >10000 to >1FFFF

This should be enough information to absorb for this tutorial.

So as not to leave you with nothing to play with , here is another high-speed command to add to your graphics routines. This is the PSET command, which can draw a dot in the VRAM or XRAM. A logical operation is done on the colour of a dot that is already displayed.

```
PSET     DATA PSETWS,PSET1
PSETWS BSS    >20
PSET1    MOV  R13,R9
         MOV   *R9+,R1          get x
         MOV   *R9+,R2          get y
         MOV   *R9+,R3          get colour
         LI     R8,>2D00        mxd select vram (00)
         BLWP  @VWTR
         SWPB   R1
         LI      R0,>0024       get x low
         MOVB   R1,R0
         SWPB    R0
         BLWP    @VWTR
         SWPB   R1
         LI      R0,>0025       get x high
          MOVB  R1,R0
        SWPB    R0
          BLWP  @VWTR
        SWPB   R2
          LI      R0,>0026       get   y low
```

```
          MOVB    R2,R0

         SWPB    R0
           BLWP    @VWTR
           SWPB    R2
         LI      R0,>0027      get y high
         MOVB    R2,R0
           SWPB    R0
           BLWP    @VWTR
         SWPB    R3            get the colour
         LI      R0,>002C
            MOVB    R3,R0
           SWPB    R0
            BLWP    @VWTR
            LI      R0,>2E50      the 5 is the pset command, the 0 is
where to put logical op.
            BLWP    @VWTR
*
            LI      R0,>0F02      point at status reg. #2
            BLWP    @VWTR
            CLR     R0
PSETST      MOVB    @VDPSTA,R0
            ANDI    R0,>0100      check CE bit in S#2
            JNE     PSETST        if not reached command end
then wait
            LI      R0,>0F00      get our usual TI status
            BLWP    @VWTR
            RTWP
```

If you add this to the last program, then you can draw dots on the screen by loading R0 with the x co-ord, R2 with the y co-ord and R3 with the colour. This command is so fast that it doesn't really need to check status register #2 as it runs faster than the 9900. I have not tried to do this with the SGCPU card operating in 16 bits. With the LINE and the PSET commands you should now be able to start experimenting with graphics and getting some nice screen displays.

In the next tutorial I will start to try to explain logical operations and add

BISECTION REVISITED
(by Francesco Lama)

Lost in the mists of TI*MES, or, to be precise, in issue 36, I presented a presented for finding the zeros of an expression of the type f(x,y) by using a numerical method called BISECTION. The original program was written in Extended Basic and may have appeared to some of you who tried it out rather slow. The C99 listing below is almost identical in layout to the original program, but is written in C99. Most of the lines are commented, so it should be easy to check how a BASIC routine can be translated into C99. When compiled this program runs up to 6 times faster than the Extended Basic one.

Besides bisect(a,b,c,d) I am also supplying a main() which is essential to running the subroutine and a trial function (appearing after bisect).

Together they form a fully working demo program for bisect. Try running it for xa=0.33 and xb=2 and require a precision e=0.000001. The result should be very close to 1.

In order to compile and run this program follow the same procedure I have given in all my previous C99 articles in any recent issue of TI*MES.

```
/* demo programme for bisect(a,b,c,d) enables one to compute the */
/* zeros of the user defined function func(x,y) */
/* up to the maximum precision availble of 10 significant figures */
#include dsk1.prf
#include dsk1.stdio
#include dsk1.floati

main()
{
 int j;
 char *c,s[16],s1[16];
 float x[8],xa[8],xb[8],e[8];

 printf("\n lower limit of interval xa,xb ");
 c=fpget(s,xa);
 printf("\n upper limit of interval xa,xb ");
 c=fpget(s,xb);
```

```
 printf("\n enter precision ");
 c=fpget(s,e);
 bisect(xa,xb,e,x);

 c=ftos(x,s1,0,0,0);
 printf("\n %-16.16s \n",s1);
}

/* the function bisect(a,b,c,d) allows one to find the zeros of a user */
/* specified function func(x,y) within the interval a,b and with preci-
sion */
/* c (c>b-a) (b>a). note that this program will only run as a subroutine
*/
/* in C99. the program calling it will have to have include statements
at */
/* the beginning for floati and stdio. moreover a user defined function
*/
/* must be provided, and when loading and running the final program
*/
/* FLOAT;O and CSUP must be loaded before the program object file.
*/

 bisect(xa,xb,e,xp)
 float xa[8],xb[8],e[8],xp[8];
  {
   int i;
   char *c,s[16];
   float xba[8],ya[8],yb[8],y[8],yo[8],yyo[8],f[8];
   float r0[8],r1[8],r2[8],r3[8],x[8];

 if(fcom(xa,">",xb))
   {
    c=fcpy(xa,r2);
    c=fcpy(xb,xa);          /* swap */
    c=fcpy(r2,xb);
   }

 i=0;                  /* r0=0 r1=1 r2=2 */
 c=itof(i,r0);
 i=1;
```

```
  c=itof(i,r1);
  i=2;
  c=itof(i,r2);


  c=fexp(xb,"-",xa,xba);  /* compute length of interval xa,xb */
  c=fcpy(xb,x);          /* set x=xb before calling sin() to prevent */
  func(x,f);            /* xb being altered on return */
  c=fcpy(f,yb);          /* yb=f(xb) */
  c=fcpy(xa,x);          /* the same as the above 4 lines is repeated for */
  func(x,f);            /* xa */
  c=fcpy(f,ya);          /* ya=f(xa) */

  c=fexp(ya,"*",yb,r3);
    if(fcom(r3,">",r0))   /* check for error condition generated by the */
      {                  /* function having the same sign at xa and xb */
       puts("ERROR! function not zero in xa,xb!");
       goto lab3;
      }

lab1:              /* first loop of bisection algorithm starts here */
  c=fcpy(f,yo);     /* yo=f (y old is given the present value of f) */
  c=fexp(xa,"+",xb,xp);
  c=fexp(xp,"/",r2,xp);
  c=fcpy(xp,x);       /* x=xp as above to prevent xp changing on return
*/
  func(x,f);          /* calculate func() result in f */
  c=fcpy(f,y);
   c=fexp(y,"*",yo,yyo);    /* yyo=y x yo the product of new and old z */

     if(fcom(yyo,">",r0))  /* if yyo>0 then */
       {
        c=fcpy(xp,xa);     /* xa=xp */
        goto lab1;         /* return to beginning of loop 1 */
       }
                    /* otherwise */
   c=fcpy(xp,xb);          /* xb=xp */

lab2:              /* start of bisection loop 2. The following 9 lines are */
  c=fcpy(f,yo);  /* identical to the ones in the previous loop */
```

```
  c=fexp(xa,"+",xb,xp);
  c=fexp(xp,"/",r2,xp);
  c=fcpy(xp,x);      /* x=xp as above to prevent xp changing on return
*/
  func(x,f);         /* calculate func() result in f */
  c=fcpy(f,y);
   c=fexp(y,"*",yo,yyo);   /* yyo=y x yo the product of new and old z */

     if(fcom(yyo,">",r0))  /* if yyo>0 then */
      {
       c=fcpy(xp,xb);    /* xb=xp */
       goto lab2;        /* return to beginning of loop 2 */
      }
                  /* otherwise */
   c=fcpy(xp,xa);        /* xa=xp */

   c=fexp(xb,"-",xa,xba);   /* recalculate length of xa,xb */
     if(fcom(xba,">",e))   /* if xba>precision then */
       goto lab1;        /* return to the beginning of loop 1 */

lab3:                    /* return begins here */

   return xp;     /* return value of asi to calling programme */
  }            /* END OF insc(a,j,b) SUBROUTINE */

 func(x,f)        /* user provided function */
 float x[8],f[8];  /* f=2-x-3timesx^2+x^3 */
 {
  int i;
  char *c;
  float c0[8],c1[8],c2[8],c3[8];

  i=2;
  c=itof(i,c0);
  i=-1;
  c=itof(i,c1);
  i=-3;
  c=itof(i,c2);
  i=1;
  c=itof(i,c3);
```

```
c=fexp(c3,"*",x,c3);
c=fexp(c3,"*",x,c3);
c=fexp(c3,"*",x,c3);

c=fexp(c2,"*",x,c2);
c=fexp(c2,"*",x,c2);

c=fexp(c1,"*",x,c1);

c=fexp(c0,"+",c1,f);
c=fexp(f,"+",c2,f);
c=fexp(f,"+",c3,f);

return f;
}
```

Richard Twyning Writes

14th TI Tref in Stuttgart

I can confirm that the Tref will be on the 1st, 2nd, and 3rd of October 1999.

I don't have any details about where it will be held, or hotels etc., only dates, but watch this space.

One important date for your diary is the AGM which will be held at the usual place.

Here are the details again for those who are unsure...

The St. John's Ambulance Hall,
Trinity Street,
Derby.

Saturday 10th April 1999.

See you there.

Once again, apologies for such a short article, I promise I'll try and write more for the next issue.

General Secretary over and out!

TI-Artist format pictures explained
by
Bruce Harrison

Here's how the files are organized. There are two files, the one ending in _P having character patterns, and the one ending in _C having the color information. Neither file has a header. Each has a content part of 24 sectors in memory image (aka program) form, plus a 25th sector that's the directory sector.

The _P file contains 768 character definitions of eight bytes each. Thus the first eight bytes define the character (8 pixels wide by 8 high) in the upper left corner of the screen. The next 8 bytes define a character pattern on that same row of the screen just to the right of the upper left corner. They continue in like manner across that first row (8 pixels high) of the screen for 32 patterns in that row, then the next group of 32 character patterns form the second 8-pixel high row on the screen.

This continues for 24 rows to make the complete screen full of character patterns.

The _C file defines the colors, again in groups of 8 bytes starting at the upper left corner of the screen. In this case, each byte defines the colors for just one pixel row of a character pattern. Each byte is divided into two nybbles (4 bits each) in which the high order nybble defines the foreground color for the corresponding 8 pixels from the patterns, and the low order nybble defines the background color for that byte of the pattern. The second byte of the color file defines colors for the second pixel row of the upper left corner pattern. Thus the groups of 8 bytes in the color file correspond to the groups of eight in the patterns file, defining in each 8 bytes the colors for an 8 by 8 pixel block.

The color coding follows the Editor/Assembler scheme, starting with 0 as meaning transparent, 1 meaning black, etc. continuing on to 15 meaning white. In other words, the codes for the colors are just one less than those given in the Extended Basic book.

TI-Artist file formats
by
Chris Faherty

Two files are used NAME_P & NAME_C. Each one is 6144 bytes in length and corresponds to a binary dump of the VDP memory for the pattern table (_P) and the color table (_C). These are described in the 9918 manual and also the E/A manual I believe. As far as Instances and Fonts, their format is described in the back of the manual. They are text files containing comma-delimited decimal numbers.

Something to the effect of:

Instance:

width,height
pattern data

Font:

Basically a bunch of Instances concatenated, but the first couple of lines is different.

char
width,height,horskip
pattern data

The Font and Instance formats don't support color and the pattern data is the same as you would find in VDP memory. i.e. an 8x8 matrix is defined in successive 8-byte groups.

TIA Plus added a Movie format and a Vectors format. Not sure if these would be that useful.. and I would have to dig up

the source to figure them out as well :)

I am out of town at the moment so I hope I haven't made any mistakes with the formats. I also use Linux very often so I am glad to hear of your project!

/* Chris Faherty <chrisf@america.com> */