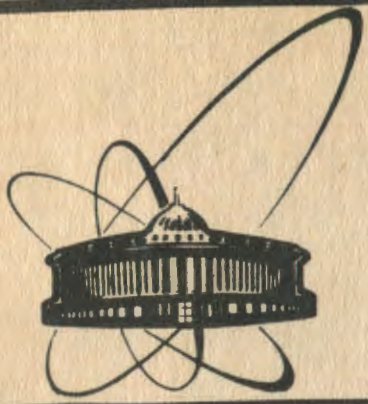

сообщения
объединенного
института
ядерных
исследований
дубна

E10-89-302

V.E.Zhiltsov

LIVE HISTOGRAMS IN MOVING WINDOWS

Example of implementation

1989

1. INTRODUCTION

Real world is arranged so that very often events can not be observed directly. It is always so in high energy physics. Therefore we have to apply some detector, some hardware to read data out of the detector and - as a rule - a computer with data processing and data presentation software.

In an ordinary modern high energy physics experiment there are usually dozens, or even hundreds of detectors of arbitrary complexity. Evidently, all of them should be carefully tested and tuned so that the experimental setup could produce trustable data. Of course, any detector should be tested with some testing equipment before it is installed in the setup. Such an equipment is also needed for research of the detectors.

Testing is usually being done by applying some source of test signal (natural or artifical) to the detector. With the help of data taking and presentation hardware and software one may compare the data with what is expected and tell what the problem is with.

The testing software should meet the following requirements:

- it should work with the same hardware as that used in the experimental setup: there´s often no other hardware;
- it should provide the user with possibilities of hardware control and data presentation not necessarily needed during the experiment: some things are of interest for test purposes only;
- user interface should be as simple as possible, and data should be processed on the fly and displayed in some easily uderstandable form: one needs not to eat the whole egg up to know it is spoiled;
- hardcopy of the test results should be of quality adequate for passportisation, human communication and publication purposes: easy-to-read and fine-looking operator´s journal may help to save a lot of energy;
- computer environment it works with should be compact and simple to operate: a student or two may always be found to do the dull job.

An example of software for testing some specific detector is discussed here. The detector under consideration is the so-called multi-wire proportional chamber.

2. HARDWARE

Most popular detectors used now are position sensitive detectors which are usually multiwire proportional or drift chambers. Such a chamber is a thin box up to several meters long and high, and about fifty or so millimetres deep with up to several hundred thin wires stretched inside it. It is filled with a special gas mixture, and high voltage of several kilovolts is applyed to it. Each wire is connected to its own amplifire and some other electronics. As a charged particle goes through the chamber, it ionizes the gas, and the appropriate amplifire senses that local charge and produces an electrical pulse. We can tell position of the particle on the hit wire number.

In particular, the chamber under test is 1.5 meters long, 1.0 meter high and 50.0 millimetres deep. It contains 640 wires which are stretched horizontally. It is developing at JINR/1,2/ and is equipped with proprietary read out hardware in CAMAC standard. The computer used is MS-DOS IBM PC/XT equivalent with 256 KB of RAM, two 5.25" 360 KB floppy disks (only one is needed actually), CGA card and EPSON-like dot matrix printer. This environment is quite compact and simple to operate.

3. GENERAL PROGRAM DESCRIPTION

The program utilizes the so-called "desk-top methaphor". That means that from the user´s point of view it looks like a stack of 12 windows he interacts with the program through (see fig1). The windows may be reordered, moved around the screen, changed in sizes and contents. Any action is possible with the window that is on top of the "stack" only. Any window may be made the top one - that is, the "active" one. The contents of a window can be changed by the user or by the data taken from the hardware. The interaction with the program reminds that of an ordinary screen text editor. If, for example, the user wants to change some parameter, he has just to enter the appropriate window and press the appropriate keys. Any text may be placed to any window (with some minor exeptions discussed below).

Not all the windows were created equal. There are 8 graphical windows to display histograms and 4 text windows to display some text. There is also one more "virtual" window - the 13th one - which exists only when active. Some service functions are performed with the help of this window. Two text

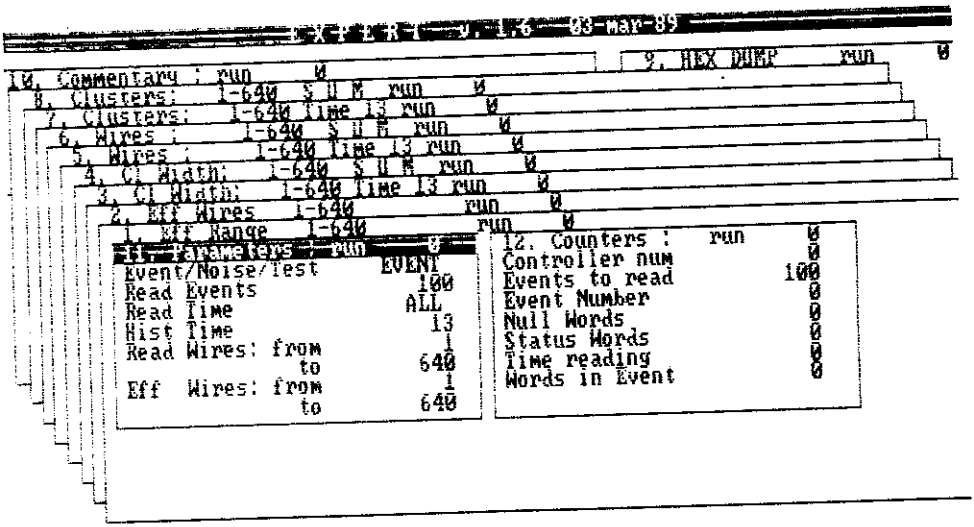

Fig.1.

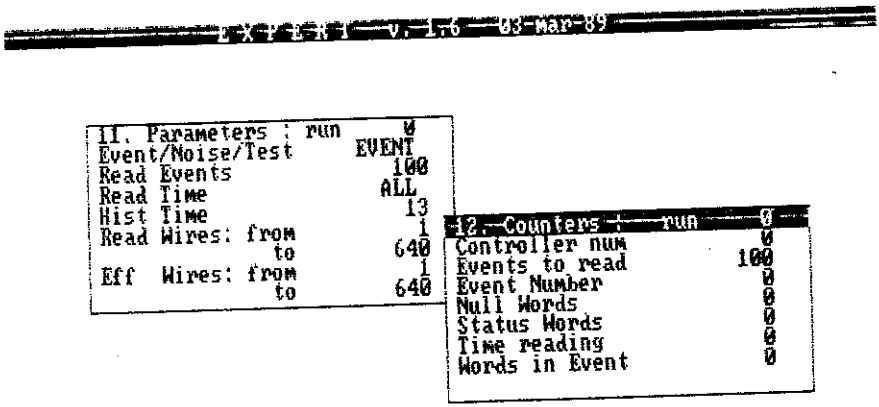

Fig.2.

windows are somewhat special ones. These are the "PARAMETERS" and the "COUNTERS" windows (fig.2). The user can not change the contents of the "COUNTERS" window, and only valid in a certain sence characters may be put to "PARAMETERS" window. Size of neither of those windows may be changed also. There is no need for this.

The two other text windows are the "COMMENTARY" and the "HEX DUMP" ones. The "COMMENTARY" window acsepts any text and can be of any size up to the whole screen. No hardware data

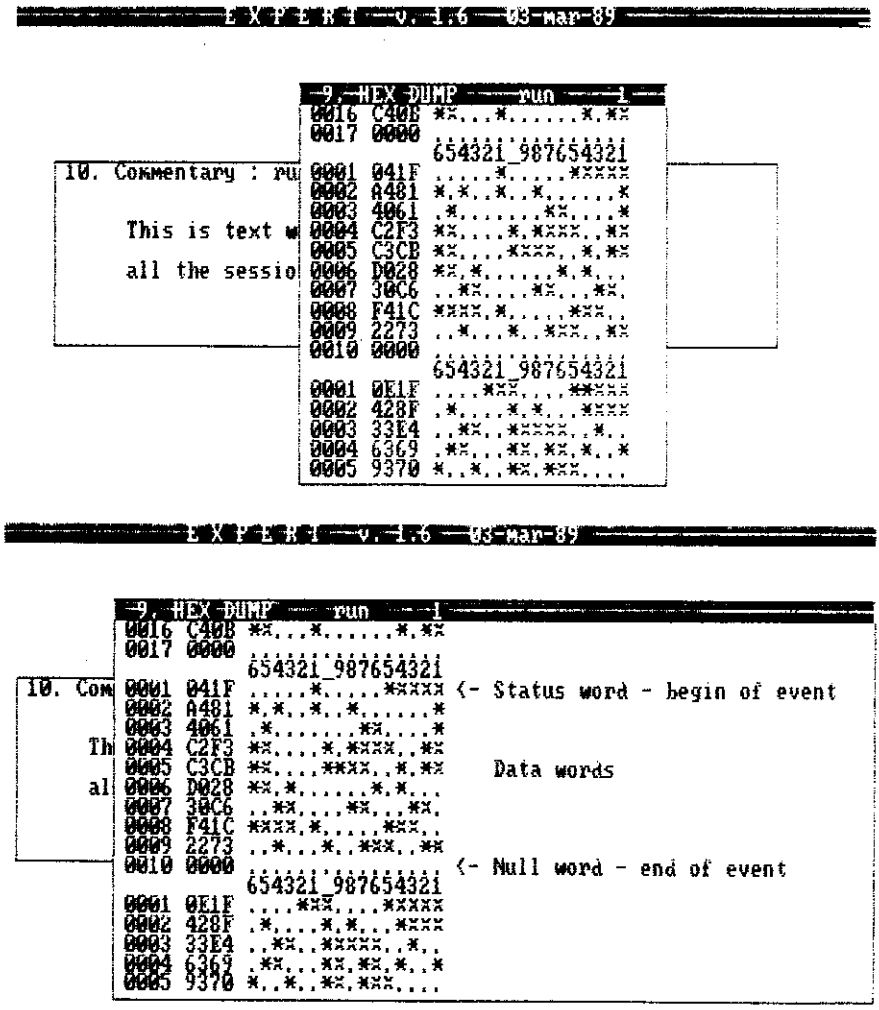

Fig.3.

can change the contents of this window. It exists all the session through. The "HEX DUMP" window is just what it says: the hexadecimal dump of raw data as they are taken from the CAMAC hardware. The text scrolls within this window as every data word is read out. This is useful to fix hardware problems which may be marked with plain text to communicate to hardware personell (fig.3).

In eight graphical windows eight different histograms are represented. The histograms are of fixed size each but the

user may change the scale (pan and zoom) to see any part of any histogram (fig.4). Those windows may be changed in size from a certain minimal size up to the whole screen. If all the eight windows are of minimal size, they may fit the screen so that they all are seen with no overlapping (fig.5). The histogram in the window that is currently "active" is changing on each data word read from the CAMAC hardware so that one may observe the data taking process while it is in progress. The

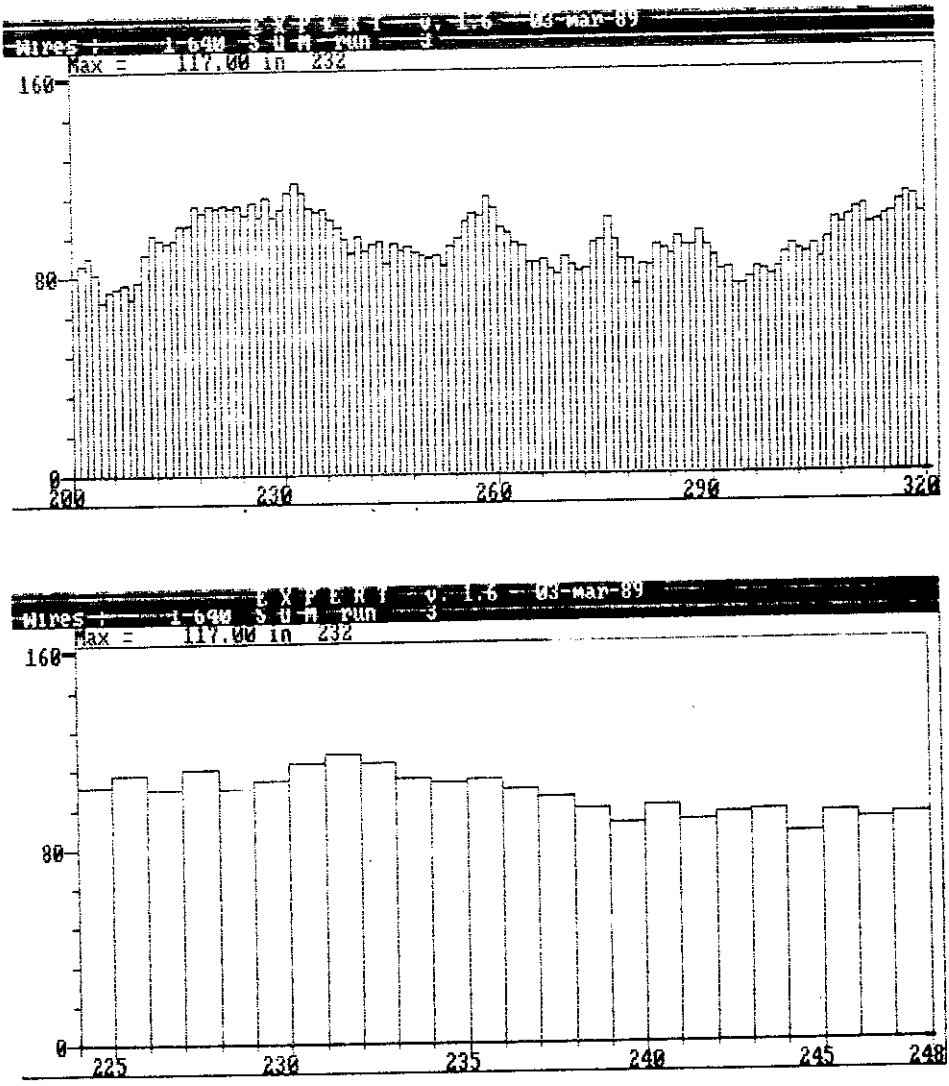

Fig.4.

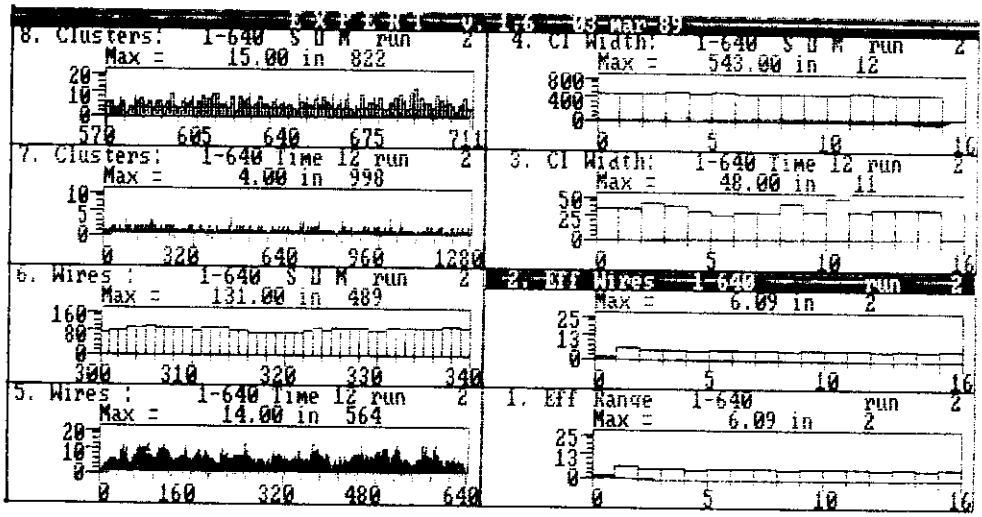

Fig.5.

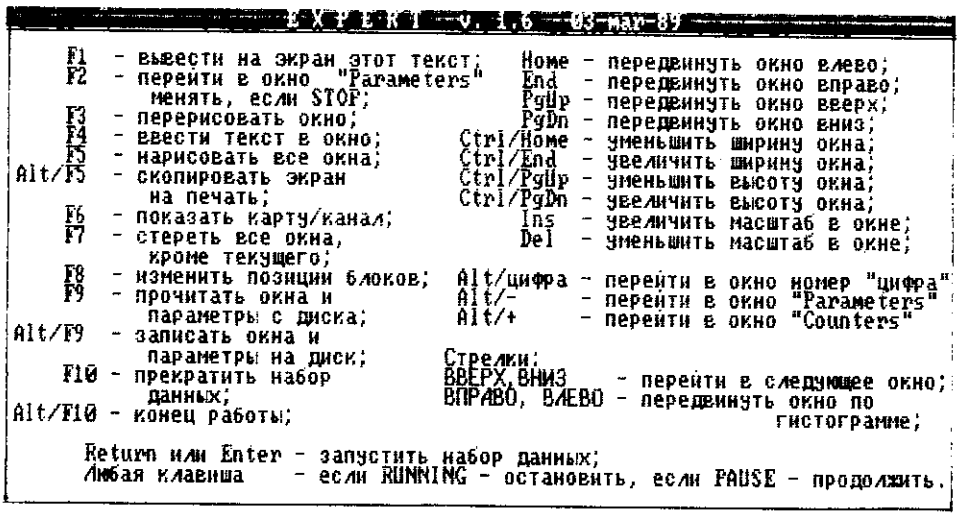

Fig.6.

user may browse through the windows quickly to decide whether there are any problems with the object under test. After the test is done the windows may be configured on the screen in any suitable manner, a commentary text may be put to any graphical window, and the hardcopy may be produced.

Naturally, there is some "help" facility. Help is implemented in two ways. First, there is a prompt line on top of the

screen specific for each mode the program is in. The current mode is indicated and a short reminder of what key may be pressed is displayed in it. Second, the full list of the control keys and their functions descriptions may be displayed at anv time on the whole screen on pressing the "help" key (F1 key). Of course, it may be hardcopied as it is shown in fig.6. Practically, this is all that is needed to know the full program control.

All the parameters settings and the screen window arrangement may be saved onto disk for later sessions.

4. SOME PROGRAM INTERNALS

The windowing technique utilized here is sometimes called "RAM-screen". To move a window we have to copy the appropriate part of the screen - the window - to some other place in the screen memory and restore the part of the screen that has been discovered after that. So it has to exist somewhere. Best of all is to keep the whole screen that underlies the moving window in program data memory - in RAM.

As we have many windows and each one is to be moved, we have to keep all of them in RAM, also, as single windows. So if, for instance, there are M windows and some window N is to be moved, the following preparations should be undertaken:

```
Store the current window to RAM window area;
Clear the screen;
Restore M-1 windows from RAM window area,
excluding window N;
Copy screen to RAM screen area;
Restore window N from RAM window area;
```

In fact, this is a "Go to Window N" procedure. Restoring many windows is time-consuming. As we usually go to window other than the current one, we may speed up the process by reducing it to just:

```
Store the current window to RAM window area;
Copy screen to RAM screen area;
Restore window N from RAM window area;
```

To restore a window and process it properly some information is needed in addition to its contents. Thus, to draw a histogram, its values should be converted to screen points to fit the window according to the formulae:

```
ScreenValue : = WorldValue * Coeff1 + Coeff2;
```

Where the conversion coefficients coeff1 and coeff2 must be evaluated for both horizontal and vertical dimensions (X and Y) of the window,according to window location and size and maximum and minimum histogram values:

```
Coeff1x : = (ScreenX2 - ScreenX1)/(WorldMaxX - WorldMinX);
Coeff1y : = (ScreenY2 - ScreenY1)/(WorldMaxY - WorldMinY);
Coeff2x : = ScreenX1 - WorldMinY * Coeff1y;
Coeff2y : = ScreenY1 - WorldMinY * Coeff1y;
```

Where ScreenX1, ScreenY1 and ScreenX2, Screen Y2 are screen coordinates (in screen points) of upper left and lower right corners of the window, respectively. WorldMinX and WorldmaxX are the lowest and highest bin numbers of the histogram to draw. WorldMinY and WorldMaxY are the minimum (always zero) and maximum bin values of it. The coefficients under consideration are to be reevaluated each time the window moves or the histogram limits change. It occurs, for instance,if the histogram maximum overflows the current window vertical size.

Hence, we have to define the window identificator, the window screen coordinates, the histogram limits, pointer to the histogram array and a few other things for service purposes such as, for instance, a flag to indicate that the histogram is already drawn. So, a window appears to be a structure (as it is in Pascal):

```
Windiow = record
            WindowID                    :integer;
            Header                      :Textstring;
            Sx1, Sx2, Sy1, Sy2          :integer
            Wx1, Wx2, Wy1, Wy2          :integer;
            Histogram                   :HistPointer;
            ...
            Drawn                       :boolean;
          end;
```

All the windows are defined as an array of such structures:

```
Windows : array [1..MaxWindow] of Window;
```

Also, information about window ordering is needed. It is kept in the window layout array:

```
WindowLayout : array [1..MaxWindow] of integer;
```

This array contains window identificators (which are window numbers) and is used in the following sence: WindowLayout[1] is the active window number, WindowLayout[2] is the window beneath the previous one, and so on. The array elements are reordered each time the active window number changes. There are two ways to change it. The user may "go" to window next to the current one ("below" or "above") or select a window by its number. The "above" window is the one that is the "lowest" in the screen window stack. Its number is WindowLayout[MaxWindow].

Any action with a window is fulfilled according to the following scheme:

```
with Windows [WindowLayout[1]] do
begin
  ...
end;
```

The idea of windowing mechanism discussed so far has been derived from Turbo Graphic Toolbox package from Borland International /3,4/. A few routines have been modified and some routines have been added to implement the following functions:

- scroll window content within the window borders;
- write scrolling test to window;
- invert rectangular area within the window;
- move the inverted area within the window;
- display and move text cursor within the window in graphic mode;
- draw a single histogram bar (erase existing bar and draw a new one - to make histogram live);
- draw histogram axis and histogram;
- change window size (horizontal and vertical dimensions separately);
- change the displayed portion of the histogram (pan and zoom with appropriate change of histogram axis);
- change active window number (so that to update all window attributes);

Some optimization was done to draw a single histogram bar fast enough. Only points that are not drawn are fired up - a check is made if the point is already drawn. As the new bar value is usually higher than the old one, the bar is drawn down till first drawn point is met. If a bar is "narrow", i.e. if it is to be displayed as a single line - one line is drawn only. Erasing is performed likewise - drawn points of the old bar are erased only, and after the new bar has been drawn (fig.7).

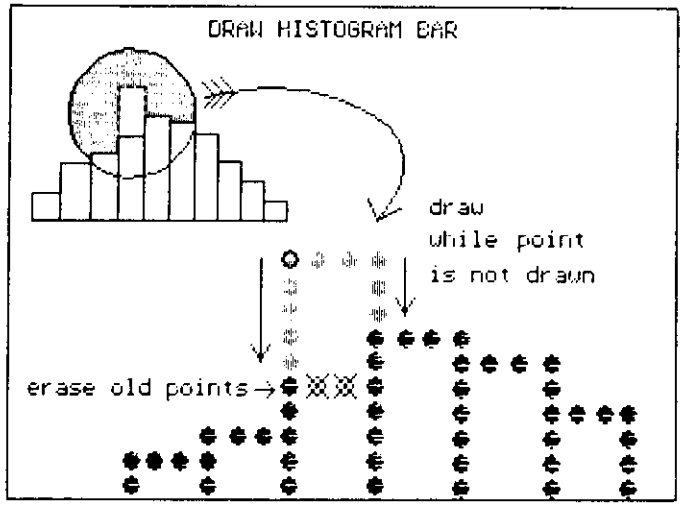

Fig.7.

The main cycle of the program is clear and simple and descriptive enough to understand the functions of all the control keys:

```
...
(Initialization part)
...
repeat
begin
  if ord (CtrlChar) <> ESC then read (Kbd, CtrlChar);
  if ord(CtrlChar) = ESC then read (Kbd, CtrlChar);
  case ord (CtrlChar) of
    Alt1..AltE : GotoWindow (ord (CtrlChar) - 119);
    DownArrow  : NextWindow(-1);
    UpArrow    : NextWindow(1);
    LeftArrow  : MoveHist(-1);
    RightArrow : MoveHist(1);
    EndKey     : WindowMoveHor(1);
    HomeKey    : WindowMoveHor(-1);
    CtrlEnd    : WindowZoomHor(1);
    CtrlHome   : WindOwZoomHor(-1);
    PgUpKey    : WindowMoveVer(-8);
    PgDnKey    : WindowMoveVer(8);
    CtrlPgUp   : WIndowZoomVer(-8);
    CtrlPgDn   : WindowZoomVer(8);
    InsKey     : ZoomHistogram(1);
    DelKey     : ZoomHistogram(-1);
```

```
      F1          : Xhelp;
      F2          : ChangeParameters;
      F3          : RedrawHistogram;
      F4          : TextToWindow;
      F5          : RedisplayAll;
      AltF5       : PrintScreen;
      F6          : ShowBinValue;
      F7          : ClearScreen;
      F8          : ChangeCamacN;
      F9          : LoadWindows;
      AltF9       : StoreWindows;
      F10         : StopRun;
      AltF10      : Quit  : = true;
    end;
    if Running or (ord(CtrlChar)=ENTER) then TakeData;
  end; until Quit;
```

Data reading cycle is hidden in TakeData subroutine. It can be seen that it is activated on pressing ENTER key. If any key is pressed during the data reading cycle, the cycle is suspended and program control is transfered to the outer cycle - the main program cycle - to see what particular key has been pressed. After the appropriate function has been accomplished the data reading cycle is resumed. Data decoding, histogram filling and displaying (with "Draw Histogram Bar" routine) is being performed during it. So, if the user is sitting quietely staring at the screen he may see the process in progress, if the currently active window is one of the histogram windows or the "HEX DUMP" window or the "COUNTERS" window - i.e. the one which content can be changed by the data. Or he may switch to any other window, or move the window, or change its size - the process will be going on untill the predefined (by the user) number of events is reached or F10 key (the "Stop Run" command) is pressed.

5. CONCLUSION

Computer graphics adds new dimension to man-mashine interaction. It especially is true if you deal with complicated data aquisition hardware. If it is the case then, as experience shows, even simple, monochrome, two-dimensional graphics can save a lot of time and labour.

The language the program is implemented in is Pascal. Some portions of routines that deal with windows and RAM-screen are implemented in Assembler for speed and efficiency. So, we may conclude that Pascal is not an ideal langauge for graphics (and data acquisition).

Assembler is hardly an ideal language for anything that is more than memory-to-memory move. There is C programming language, of course, that seems to be suitable for almost everything. But - what we deal with if we do graphics are images or objects. So an object-oriented approach might be reasonable.

We now see a dramatic pace of computer graphics progress: raw power of computers increases, specialized chips appear, new architectures emerge, new software appears - algorithms, languages, standards, and so on. Along with all that prices steadily go down - we are getting still more and more bang for the buck. State-of-the-art computer graphics has come from separate rooms to under the table, and is now taking its place on top of the table for table-top prices. So, one may expect that in near future compact and simple to operate test systems will have more graphics of higher resolution and colour. It will bring to life better detectors, those will let us learn more about the real world, and that will stimulate new ideas on computer graphics to appear.

The author is grateful to D.A.Smolin for helpful discussions on CAMAC hardware.

REFERENCES

1. Kiryushin Yu., Vishnevsky A. - Nucl. Instr. and Meth., 1984, A252(2,3), p.281.
2. Vishnevsky A. - JINR Commun. 13-83-15, 1983.
3. Turbo Pascal Reference Manual. Borland International Inc., 1985.
4. Turbo Graphics Toolbox Reference Manual. Borland International Inc., 1985.

Received by Publishing Department
on April 28, 1989.