Bull. Nov. Comp. Center, Math. Model. in Geoph., 8 (2003), 19–25
© 2003 NCC Publisher

User interface of the control program for mobile data gathering systems

Pavel R. Andreyev

Basic principles and engineering solutions of the AXON's user interface are described in this paper. The software AXON is a part of the mobile geophysical data gathering systems "Rosa-D", "Rosa-N", "VIRS-M" developed in the ICM&MG. AXON provides the automatic and manual (human supervised) modes and contains a kit of tools for diagnostic and adjustment purposes.

Interface overview

The interface described in this paper has been designed for the MS-DOS compatible operating system (OS). Though the DOS is now considered to be outdated, it is still working in certain areas, where more powerful OSes (Windows, Unix, QNX) could hardly be used due to technical or economical reasons. Say, the data gathering systems developed [2, 3] do not require a high-performance computer (even 80486-based computer can be used), which significantly reduces the system costs. Also, because the DOS ideology is simple and widespread, training the staff maintenance of the data recording system becomes easier.

Unlike modern operating systems containing both textual and graphic interfaces ready-to-use, the MS-DOS, virtually, contains neither. Standard C (and C++) library oriented to perform streamed input/output also includes no routines convenient enough to organize user interaction. As a result, DOS programs have to bear their own implementations of interface. Many of them built on the ground of two widespread libraries supplied together with Borland's compilers – Turbo Vision to organize text-oriented interaction with the user, and the BGI (Borland Graphics Interface) to use computer graphic capabilities. There is a variety of other libraries available, both original and enhancements (for example, graphical extension of the Turbo Vision). This fact was taken into account when the AXON software was under construction, but products known to the developer happened to be unsuitable and as a result the original library named IMAGINE has been written. It may be also called an API (Application Programming Interface) to be used to make program user interface. Both AXON and IMAGINE have been written in C++ and compiled with Borland C++ v3.1 compiler.

The main features of IMAGINE are the following:

- Unified API for text and graphic video modes;
- 256-color video modes support (VGA and VESA);
- 2.5D style support (quasi-3D graphics);
- Mouse support in VESA graphic modes;
- Russian alphabet characters can be used in VESA graphic modes (built-in character generator); and
- Fast video output and transparent graphics acceleration routines.

Unlike majority of products of the class that have different APIs for text and graphics, IMAGINE has a unified input-output routine set consisting of several call groups:

- "Keyboard" – keyboard data input;
- "Mouse" – interaction with the mouse driver and the mouse cursor control;
- "Text output" – single characters and strings output, pseudographic frames and borders drawing, text-mode windows drawing;
- "Graphics output" – lines and rectangles drawing, rectangle filling operations.

All the functions produce the same results independently of the current video mode save that IMAGINE ignores graphics calls in the text video modes. This approach enables the programmer to concentrate on the screen design while the library performs all the low-level work on the video card interaction. Additionally, the replacement of the video mode used does not lead to malfunction of the IMAGINE-based programs because of the library checking procedures revealing attempts to access the areas outside of the screen.

1. Supported video modes

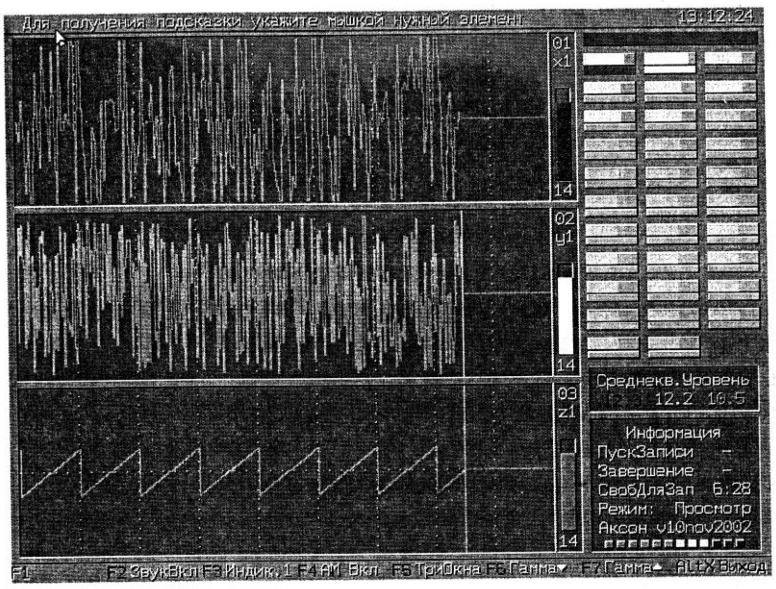
IMAGINE manages five video modes altogether, which, on the one hand, ensures the functionality high enough and, again, the rejection to support obsolete video modes allows optimization of the speed of video output algorithms:

- `0x003` (VGA), 80×25×16 colors text mode. This is a standard text video mode for most of the DOS applications;
- `0x108` (VESA), 80×60×16 colors text mode;

- `0x013` (VGA), 320×200×256 colors graphics mode. Considered as 40×25×256 text mode by the IMAGINE text output functions;
- `0x101` (VESA), 640×480×256 colors graphics mode. Considered as 80×30×256 text mode by the IMAGINE text output functions;
- `0x103` (VESA), 800×600×256 colors graphics mode. Considered as 100×37×256 text mode by the IMAGINE text output functions.

Availability of the Russian language characters in the first three modes relies upon the Russian language driver installed in the computer system. For `0x101` and `0x103` modes IMAGINE has its own built-in character generator. In addition, the library includes several mouse support functions for the graphic video modes.

Because IMAGINE has been designed as part of the real-time software, due attention was paid to the speed of its routines. For example, all video output related functions access the video memory directly. Additionally, special optimization transparent to the programmer steps in when working in the graphic video modes, which significantly (occasionally, several times) increases the output speed by reducing overhead (video frame switching, checking for access outside of the screen, hiding mouse pointer etc.). As a result, repainting of AXON screen (containing a text and graphics in 640×480 pixels resolution) takes 0.15 s on the Pentium-166 based computer, equipped with the PCI video card. In this case, the peak output rate is better than 8 million pixels per second (figure).


Для получения подсказки укажите мышкой нужный элемент

Среднекв. Уровень
12.2 10.5

Информация
ПускЗаписи –
Завершение –
СвобДляЗап 6:28
Режим: Просмотр
Аксон v10nov2002

AXON graphics interface

2. Solutions used in IMAGINE

This section contains prototypes and short descriptions of several IMAGINE API calls. It should be noted that IMAGINE uses the data type notation elaborated by the Russian company 'ToxSoft'. The integer data types are written as `XintNN`, where X ('s' or 'u' letter) marks a 'signed' or an 'unsigned' type, respectively, and NN is the type bit width. So, the unsigned 32-bit integer (`DWORD`) is defined as '`typedef unsigned long int uint32`'.

Keyboard group

```
uint16 _fastcall KbStat(uint8 fEE=0);
```

Checks whether the code is available in the keyboard buffer; in the graphic video modes also refreshes mouse pointer position (redrawing the cursor if necessary) and, if the argument is not zero, buttons state. This allows keeping track of the mouse state transparently in the program.

Another feature of the function is Enter/Esc codes generation, when `fEE!=0` and the left or the right (respectively) mouse button is clicked. These codes will be returned by `KbCode()` function in its next call as if they were typed on the keyboard:

```
uint16 KbCode();
```

Fetches a keyboard code (waiting for it if no code available). Unlike standard C functions, there is no need to check the value returned and to make the second call for obtaining the extended code. The ASCII codes are returned in the low byte (the high one is zero) and extended codes – in the high byte (the low is zero in this case). Return values are available to the caller hereafter through the global `KbdKey` variable.

Text output group

```
void _fastcall TxChar(uint8 Sym=7);
```

Displays a character. This function is basic and used by all other functions in the text output group. The function considers the screen surface as a two-dimensional array and the argument character will not be displayed if it does not fit the array boundaries. The code `0x0A` (LF, Line Feed) in the lowest screen row is processed in a special way - the scrolling operation is performed, that is, all contents of the screen is moved up one row losing the topmost row and the lowest one is cleared. In addition, `0xF4:0xF7` character codes select the corresponding color mask (one of the four) instead of displaying. The mask consists of two numbers specifying the background and foreground colors.

When in the graphic video mode, `TxChar()` can display characters in 2.5D-style by setting up the corresponding global variables.

```
void TxSymS(const int8 *String);
void TxBinS(const int8 *String, uint32 Value);
void TxDecS(const int8 *String, uint32 Value, uint8 Opt=0);
void TxHexS(const int8 *String, uint32 Value, uint8 Opt=0);
```

These functions perform the text string output to the screen or into the memory. The MemPrt global variable points to the memory position, where the output should be performed or, if `MemPrt==NULL`, the string will be displayed on the screen. The functions `TxBinS()`, `TxDecS()`, and `TxHexS()` are used to print binary, decimal or hexadecimal values, respectively, replacing '@' symbols found in the String argument by binary, decimal, or hexadecimal digits. As a rule, one digit replaces one '@' (`TxDecS()` has also an option to print the whole number when every single '@' found). If the amount of formatting symbols is not sufficient to receive all significant digits of the argument, only the lower digits are displayed. The `Opt` argument controls the way the resulting string will look (whether spaces should be printed instead of leading zeroes, or the plus sign be printed before a positive number, etc.). An example:

```
#include "Imagine.h"
int8 Str[16]; Str[15] = 0;
MemPrt = Str;   // Output to memory not to screen
real32 V = 7.5; // The same as float (32-bit wide)
TxDecS("Voltage=@@@.@@\n", uint32(V*100), 6);
MemPrt = NULL;
```

This example prints "`Voltage= +7.50`". Note that IMAGINE itself does not use the floatingpoint arithmetic and, therefore, is suitable to write programs for 80386-type processor without the FPU.

```
void TxFram(const int8 *Array);
```

Draws a pseudographic frame. The argument is a pointer to an array containing commands drawing the frame, and the function itself is in fact a simple language processor. IMAGINE includes two command arrays to draw simple rectangular frames (single- and double-lined). Frame dimensions are specified in two first bytes of the corresponding array and are considered as the number of macro repetitions that draw the horizontal or the vertical frame border.

```
uint8 TxWind(uint16 h=0, uint16 v=0);
```

Displays a window casting shadow. The window border includes double-lined pseudographic frame, and the shadow color depends on the color of the shadow underlying image.

```
void TxMBox(const int8 *Title, const int8 *S1=NULL,
            const int8 *S2=NULL, const int8 *S3=NULL);
```

Displays a message window in the center of the screen. The message contains up to three text strings. The window size depends on the length of a title or the longest string. All strings (including the title) are centered.

Graphics output group

```
void GrDot();
```

Displays a dot on the screen. This function is basic and used by all other functions in the graphics output group. This function considers the screen surface as two-dimensional array and the dot will not be displayed if it does not fit the array boundaries. Additionally, up to 16 viewports can be defined. Depending on the global setting, `GrDot()` will display a dot only when it is either inside of any viewport or outside of all viewports. This mechanism enables the programmer to simulate the overlapping windows structure by simple means.

```
uint8 GrGet();
```

This function returns the color of a dot on the screen and is used, mostly, by IMAGINE itself to show the mouse pointer (because many videocards perform reading from the video memory several times slower than writing). Showing the pointer, IMAGINE saves an appropriate underlying screen area in its internal buffer and then restores it back when the pointer should be hidden again. Unlike a conventional method (the usage of the AND/XOR masks), this one reduces the number of screen dot manipulations and enables the pointer to be of any color and shape. For example, IMAGINE uses 2.5D-styled pointer picture.

```
void GrPoly(sint16 x=0, sint16 y=0, uint16 h=GrXm,
            uint16 v=GrYm);
```

Draws a one dot wide rectangular frame. The frame can be displayed optional in 2.5D style by setting up the corresponding global variables.

3. AXON user interface

The AXON software has a combined interface to reach the maximum flexibility and usability [1]. The command-line keys adjust the basic settings of the recording system, which allows the data gathering and recording process to be automatically performed. The text mode window-based interface is used for diagnosing and adjustment of the system hardware. The AXON layered help system describing the command-line keys usage has been built on the same basis. Finally, the mode used most often – the Recorder – has the GUI (Graphics User Interface) made in 2.5D style. The GUI also includes a simple help system that shortly describes objects on the screen and advises the keyboard shortcuts controlling these objects.

References

[1] Andreyev P.R. Real-time software for mobile data gathering system // Proc. of the Intern. Conf. "Information Systems and Technologies", November 8–11, 2000. – Novosibirsk, 2000. – Vol. 4. – P. 19–22 (in Russian).

[2] Grigoruyk A.P. Distributed real-time data acquisition system // *Ibid.* – Novosibirsk, 2000. – Vol. 3. – P. 456–460 (in Russian).

[3] Shorokhov M.N. A vibroseismic measuring-recording station // *Ibid.* – Novosibirsk, 2000. – Vol. 1. – P. 144–148 (in Russian).