ELSEVIER

Journal of Neuroscience Methods 63 (1995) 197–204

JOURNAL OF NEUROSCIENCE METHODS

Object-oriented approach to fast display of electrophysiological data under MS-Windows™

Frédéric Marion-Poll *

INRA Station de Phytopharmacie, route de St. Cyr, 78206 Versailles, France

Received 28 November 1994; revised 5 July 1995; accepted 27 July 1995

Abstract

Microcomputers provide neuroscientists an alternative to a host of laboratory equipment to record and analyze electrophysiological data. Object-oriented programming tools bring an essential link between custom needs for data acquisition and analysis with general software packages. In this paper, we outline the layout of basic objects that display and manipulate electrophysiological data files. Visual inspection of the recordings is a basic requirement of any data analysis software. We present an approach that allows flexible and fast display of large data sets. This approach involves constructing an intermediate representation of the data in order to lower the number of actual points displayed while preserving the aspect of the data. The second group of objects is related to the management of lists of data files. Typical experiments designed to test the biological activity of pharmacological products include scores of files. Data manipulation and analysis are facilitated by creating multi-document objects that include the names of all experiment files. Implementation steps of both objects are described for an MS-Windows™ hosted application.

Keywords: Object oriented language; MS-Windows™; Fast display; Electrophysiology

1. Introduction

Microcomputers equipped with data acquisition cards replace sophisticated laboratory equipment. Fast display peripherals, large storage facilities and the combined speed and precision of analog-to-digital converters favor the trend of using microcomputers both as recorders and signal monitoring devices (Fadda et al., 1989; Turner and Schlieckert, 1990; Skeen et al., 1992). Traditional paper printouts and even oscilloscopes are no longer needed to perform experiments and to analyze data (Stromquist et al., 1990; Stys, 1991). Whatever system is used, there is a need for visual inspection of experimental data either to evaluate the quality of the recordings or to gain an overview of the results of the experiments (Fadda et al., 1989; Dietz et al., 1990).

In our laboratory, we record the electrical activity of insect olfactory and gustatory sensilla. Traditional approaches used to improve the occupation of the experimental setups by recording data on a low-cost media such as magnetic tape. Data were acquired and analyzed after the experiments, either when experiments kept a low pace or by duplicating one of the equipment necessary to replay and analyze recorded data. In order to rationalize the data analysis process, we decided to integrate our computerized data acquisition setups into a group of general purpose microcomputers running MS-Windows™ and to develop a common data analysis program. This approach improves how experimental setups are used and provides a simple way to share common resources such as storage or printing media (Turner and Schlieckert, 1990).

To this purpose, we have developed an MS-Windows™ hosted program, AWAVE. This program can read and analyze data acquisition files under two different binary formats from any computer of the network. Some of the procedures described in this paper can be implemented under most PC or mainframe environments. Implementation under MS-Windows™ by deriving standard interface objects is described, with a particular mention to the visual oriented aspect of this programming approach.

* Corresponding author: Tel.: (33) 1-30833145; Fax: (33) 1-30833119; E-mail: marion@versailles.inra.fr.

2. Hardware and software development system

2.1. Experimental procedures

Two separate data acquisition systems are used in the laboratory. Electrical activities from insect olfactory sensilla are recorded extracellularly from tip-severed sensilla (Renou, 1991; Renou and Lucas, 1994). Amplified signals (Neurolog amplifiers and filters) are converted at 10 kHz with a 12-bits resolution data acquisition board (DASH-16: Keithley, USA) driven by programs developed under ASYST (Renou, unpub.). ASYST files have a specific binary format, that includes headers and binary data. Electrical activities from insect taste sensilla are recorded extracellularly by contacting sensilla with an electrode filled with both an electrolyte and the stimulus. The electrical signal is amplified with a special-purpose DC amplifier (TasteProbe, Syntech, NL) and an RS232-programmable amplifier/filter (CyberAmp A320, Axon Instruments, USA). Data are converted at 10 kHz with a 12-bits resolution data acquisition board (DT2821, Data Translation, USA) driven by a custom DOS program (Marion-Poll and Tobin, 1991, 1992). Files generated during experiments include a header and binary data that conform to ATLAB drivers file structure (ATLAB driver, Data Translation, USA).

Typical experimental sessions generate 50–200 files a day. These data are analyzed in several steps. First, raw data are printed for archival purpose. Then spikes are interactively detected thus generating spike files. Spikes are then classed according to amplitude or waveform criteria and counted. These results are exported as ASCII data to Excel (Microsoft) spreadsheet for further analysis. The entire analysis process is performed on any microcomputer running Windows 3.11 and connected to an Ethernet network while data files remain localized either on a PC or on a UNIX server via NFS emulation. At the end of a series of experiments, raw data and results are archived on low-cost 2 Gb DAT tapes (WangDAT 3100, WangTek, Japan).

2.2. Program development environment

An AT-compatible computer was used as a development station. This computer included 16 Mb of RAM, a 640 Mb SCSI hard disk and an Intel 486DX CPU running at 50 MHz. Procedures were developed using Visual C + + (version 1.51, Microsoft, USA) and MFC classes (Microsoft Foundation Classes 2.0). Visual C + + includes a set of tools such as a source editor, a symbolic debugger, a class browser and an interface prototyper. Integrated programs called wizards, assist the process of defining the framework of the application and generate the corresponding source code. Objects developed for display and analysis of our electrophysiological data are hosted by a program under development called AWAVE.

3. Visual object-oriented programming under windows

3.1. Object oriented programming

Object-oriented programming (OOP) is intended to help programmers to create and maintain large programs. This approach is becoming available in programming languages used by the scientific community such as C + + and Pascal. One of the goals of OOP languages is to offer the possibility of creating re-usable programming objects. A programming object can be defined as a black box that exchanges messages and data with other objects. Such objects have two major properties: they are encapsulated and they can be derived. Encapsulation means that objects containing both data and procedures can restrict the visibility of their elements and hide their internal workings to the outside. Such objects communicate with other objects via messages or calls to public procedures. Their interface is completely defined so that they can be used within programs as building blocks. Inheritance is a mechanism that allows programmers to build new objects by deriving them from one or several parent objects. New procedures, parameters or messages can be added to the new object while benefiting from parents properties and procedures.

3.2. Windows messaging system and interface objects

Windows is built as a collection of objects exchanging messages. Typical objects are represented by controls such as buttons, edit controls, static text, sliders, and list boxes. These elements are functionally defined as objects with their own private data and procedures. They receive and generate messages. Windows messages have a common structure defined by an identity number associated with two optional additional parameters. Any event occurring on the screen or at the level of the peripherals (keyboard, mouse) generates messages that are directed to one of these objects. For example, when the mouse cursor is moved over a window, this window receives a continuous flow of `WM_MOUSEMOVE` messages that include the x,y coordinates of the mouse cursor. This flow of incoming messages imposes a specific structure to any program that must react appropriately (event-driven instead of procedural).

Windows interface objects are combined into higher level structures such as dialog boxes, single document (SDI) or multi-document interface (MDI) windows. Basic objects are grouped in a functional frame to present the user a set of common elements including for example a menu, scroll bars, a client area, size-modifiers, and optional toolbars. Popular word processors or spreadsheet under Windows follow the MDI structure, which means that such programs can open many documents or file at the same time. Programming such applications with non OOP languages such as C represents a difficult task. The corresponding program layout is complex and difficult to maintain.

3.3. C + + language: classes and derivation

C + + is considered as an OOP language in that the basic mechanisms of encapsulation and derivation are implemented by classes. Classes represent an evolution of 'structures' defined in C, i.e., collections of variables of different types. In addition, they include a collection of methods or procedures that work on data defined within or outside the class. Procedures and data can be declared either as public or private.

One of the most useful mechanism of C + + is the possibility of deriving existing classes. For example, a typical C + + library contains classes to manage dynamic arrays of integers, pointers or different objects. The behavior of such classes can be enriched by derivation for example to implement a low-pass filter that works on the elements of such arrays. The primary task of implementing the algorithm is the same as in procedural languages but once done, the new class can be used in different contexts or further derived to add other possibilities.

3.4. Visual C + + : program framework and visual programming

Visual C + + encapsulates Windows standard interface elements within a hierarchy of C + + classes. In addition to general purpose classes (strings, arrays, lists, files), a host of classes encapsulates the user interface elements (buttons, scrollbars, menus, toolbars, etc.) integrated within a program framework. This framework structures applications around two major classes: documents and views. These classes cooperate to manage data, provide the user a representation of the data and react to user inputs. Such classes and the code-generating options of Visual C + + greatly simplify the definition and layout of a program project.

Visual C + + stands for 'visual' which means that in addition to the underlying classes and program generating facilities provided by the environment, a set of tools is available that allow one to design the user interface. The user interface includes resources such as strings, dialog boxes, menus, icons, cursors and bitmaps. For example, designing a dialog box involves 'dropping' controls within the dialog box area, defining their properties and adjusting their size and position at will. At the end of this design process, Visual C + + can be requested to generate the source code necessary to manage this dialog box. Appropriate actions in response to user input are done by trapping specific Windows messages and inserting the corresponding code. Thus, Visual C + + offers a suite of tools that allow programmers to design projects interactively, build progressively the user interface while integrating the generated procedures within a structured framework. Such programs run on different PC hardware configurations, and are portable across platforms supporting Windows services.

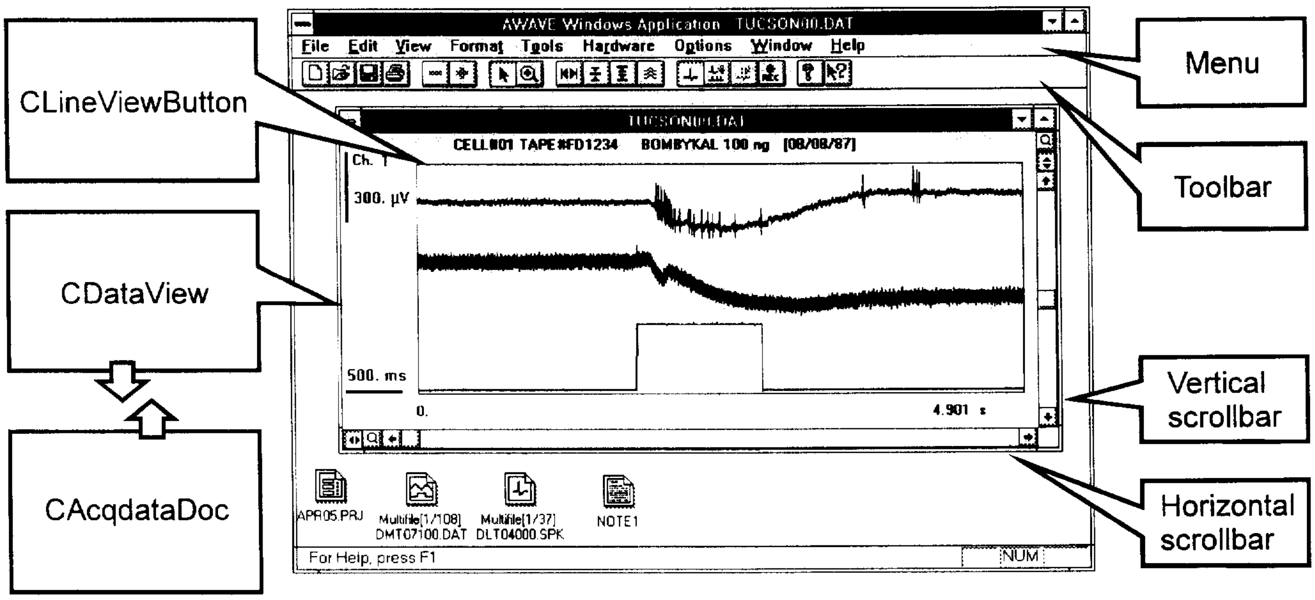

CLineViewButton
CDataView
CAcqdataDoc
AWAVE Windows Application TUCSON00.DAT
File Edit View Format Tools Hardware Options Window Help
TUCSON00.DAT
CELL#01 TAPE#FD1234 BOMBYKAL 100 ng [08/08/87]
Ch. 1
300. µV
500. ms
0.
4.901 s
APR05.PRJ
Multifile[1/108] DMT07100.DAT
Multifile[1/37] DLT04000.SPK
NOTE1
For Help, press F1
NUM
Menu
Toolbar
Vertical scrollbar
Horizontal scrollbar

Fig. 1. Interface elements of the program. AWAVE is an MDI application running under MS-Windows™. A single data file is displayed in the child window (class `CDataView`). This child window includes edit controls to display text, static controls to draw rectangles and lines, and an owner-drawn button (class `CLineviewButton`). On the bottom and right of this view, scrollbars provide a way to browse through the document (horizontal scrollbar) and to change the position and gain of the curves (vertical scrollbar). Above the client area of the application, a menu is associated to the active view, as well as a toolbar. On the bottom of the client area of the application, 2 icons represent documents inactive at that moment, respectively a text document (`CNoteView`) and another data document including a series of data files (`CDataView`). Actual data are contained within `CDocuments` objects, respectively `CNoteDoc` and `CAcqdataDoc`, which are not visible from the user.

4. AWAVE classes

Using the tools briefly depicted above, we have developed a Windows-hosted application, AWAVE, that allow us to analyze our electrophysiological data and export intermediary results to general purpose spreadsheet programs. Among the classes developed for AWAVE, three classes are essential to the structure of the application: `CAcqdatadoc`, `CDataView` and `CLineView` (Fig. 1). These three classes were derived respectively from MFC classes `CDocument`, `CFormView` and `CButton`. They cooperate to display data in a flexible and convenient way.

4.1. Data acquisition document: class `CAcqdataDoc`

Data files are managed by the class '`CAcqdataDoc`', derived from MFC's '`CDocument`' class. A `CDocument` class includes procedures to create, open, save or close a document. `CAcqdataDoc` keeps the same functionality but in addition, behaves as a data server to external objects such as `CView` which need data to build a representation of the data. `CAcqdataDoc` incorporates also ASCII data exporting procedures and basic signal processing routines.

When a data file is opened, `CAcqdataDoc` detects the structure and origin of the data file by analyzing the header (most data acquisition files have 2 parts, a header and binary data). Parameters parsed from the header are transferred into a generic data structure. Binary data are 12-bit precision words, offset binary encoded (values range from 0 to 4095, zero volt is represented by 2048). This approach ensures that new data formats can be read simply by writing the code necessary to detect the header structure and parse the appropriate parameters.

`CAcqdataDoc` makes data available to views that request them. A direct approach would involve loading in memory all data, both raw and transformed (for example by a low-pass filter algorithm). Although this approach is possible given the memory management services provided by Windows, it would waste resources and slow down all operations. We chose instead to implement a buffering mechanism so that the memory footpad of any data acquisition file is at most 128 Kb (Fig. 2). When access to original data is requested by an external routine, `CAcqdataDoc` loads as much data as possible and returns the sender a message indicating how much was loaded during this pass. Thus AWAVE can run with an acceptable response time on AT-compatible computers with modest configurations.

4.2. View: class `CDataView`

`CAcqdataDoc` documents manage data but do not deal with any representation of the data. MFC's framework proposes `CView` and derived classes as an effective ground to implement data display. `CView` classes interact with `CDocument` classes by exchanging messages. Within an MDI application such as AWAVE, views are responsible for drawing a window and all elements contained within it, i.e., curves, axes, and comments. Views also react to user inputs (menu selection, mouse movements, etc.), manage printer output, print preview, window size modifications, etc. In order to encapsulate the data display functions, we have chosen to build `CDataView` as a container to Windows controls that perform specific functions (see Fig. 1).

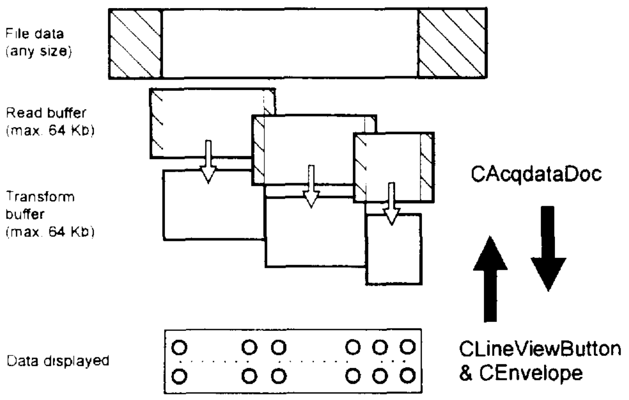


Fig. 2. Document's buffers and envelopes. `CAcqdataDoc` reads data from data files in a buffer (maximum size: 64 Kb). If transformed data are requested, for example low-pass filtered, an additional buffer is created to store the transformed data. These transformed data are computed from the raw data buffer. Additional points on each side of the raw data data buffer are usually needed to build such transforms (grey zone on each side of the raw data buffer). When the amount of data to display is larger than the buffer, data are loaded in consecutive passes. The requester (`CLineviewButton`) builds the final curve by storing the results in an additional array (`CEnvelope`) that contains the minimum number of points corresponding to the window resolution.

`CDataView` is derived from MFC's `CFormView` class. It includes a custom control to display the curves, text edit controls for the scale parameters and scale bars. In addition, two scroll-bars associated with push-buttons provide access to scale parameters (horizontal scrollbar: zoom/browse; vertical scrollbar: offset/gain). `CDataView` is associated with a menu and a toolbar including buttons that duplicate the functions of some menu items. Each control within `CDataView` can be selected and modified by the user. Parameters such as gain and signal vertical offset, horizontal zoom factor and file position can be modified either using regular menu selection, keyboard input to a selected control of the `CFormView` or with the mouse (Fig. 1). `CDataView` thus orchestrates the relations between the different elements showed on the screen, the data document and respond appropriately to Windows requests and user actions. The more specific task of displaying data is delegated to a control.

4.3. Display button: class `CLineviewButton`

While `CDataView` is constructed as a container to a group of controls, data are actually displayed by a button-derived class, 'ClineviewButton'. The design of this control is crucial to the performance of AWAVE. ClineviewButton interacts directly with Cacqdata-Doc to build a graphical representation of the data. It sends requests to CacqdataDoc and reads data in several passes. CLineviewButton provides flexible scaling on both ordinates and abscissa axes. All display operations are performed by calls to standard function of the Graphics Device Interface (GDI). This involves taking care of several aspects, vertical and horizontal scaling, and deciding how data are displayed.

Vertical scaling involves displaying each curve with an appropriate amplification factor and vertical offset. When the GDI is set under MM_ANISOTROPIC mode, this operation is limited to sending to the GDI proper scaling parameters prior to the actual drawing of each curve. Data acquisition values can then be directly fed to GDI routines. The screen coordinates of a given point are essentially defined by 4 integer (16 bits) parameters, called window and viewport Extents (wE and vE, respectively) and window and viewport Origin (wO and vO) according to the following formula:

$$S = (D - wO) * vE/wE + vO \; (1) \; with D = binary data value and S = screen coordinate$$

Independent scaling for each curve is thus achieved by modifying wO and wE just before sending the raw data points to the GDI display routines.

Horizontal scaling is not directly supported by this mechanism, essentially because abscissa (file indexes) are not 16 bits signed integers (upper limit: 32 767) but double words (upper limit: 2 147 482 647). As far as data files with more than 32 768 points are common, the basic scaling facilities provided the GDI are not enough. The simplest approach to cope with this limitation involves computing screen abscissa separately using long values and output the GDI LineTo( ) routine, pairs of points including a transformed abscissa and the binary data value.

Significant improvements in display speed can be achieved by selecting the proper GDI routines and reorganizing data in arrays before sending them to the GDI. In sample record (Fig. 2), three channels are sampled at 12 kHz during 4.9 s. Total size of the file is 354 Kb and individual data points can be indexed by a long value ranging from 0 to 58 880. Displaying each individual point of this file will involve 58 879 × 3 consecutive calls to the LineTo( ) routine. A significant gain in speed is achieved by using calls to PolyLine( ) which works on arrays of coordinates (x,y). These arrays are, however, limited to 16 384 points (under Windows™ 3.11). The complete file could thus be displayed with only 4 × 3 consecutive calls to PolyLine( ).

4.4. Display envelopes

Significant improvement of the display performances can be further achieved by taking into account the actual horizontal resolution of the output window (Fig. 3). The rationale behind this assertion is best demonstrated with real data. Let assume a horizontal resolution of 640 pixels (VGA screen) and a display window that occupies all this space. With our data sample, each channel is composed of 58 880 vectors that are drawn within 640 vertical columns (intervals). On average 92 vectors are drawn within each interval. With higher resolutions, the ratio is still important since for SVGA screens the minimal ratio is 73.6 (with 800 × 600 pixels) or 57.5 (1024 × 748). If data are displayed as interconnected lines (instead of individual dots), all vectors falling within a given interval are interconnected and the final screen output results in a single vertical line.

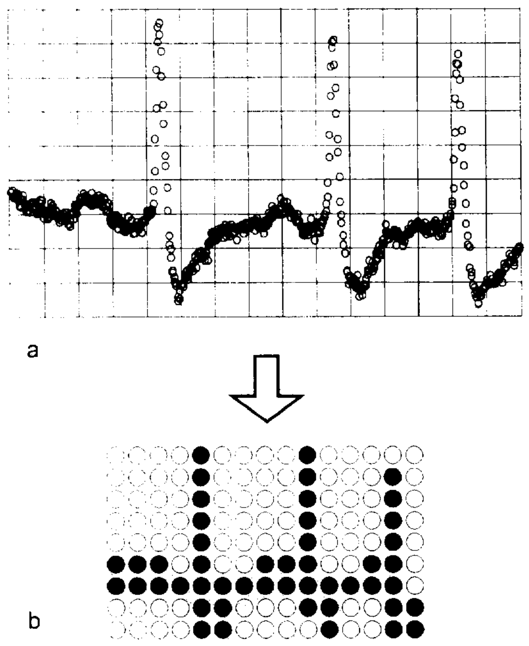


Fig. 3. Data envelope. (a) On most occasions, more data are available than pixels. In such situations, it is slower to display all vectors than to (b) build an envelope and display this envelope. An envelope is defined as an array (size = 2 × number of pixels of the window) where the maximum and minimum of all vectors are sequentially stored. Sample shows such a snipet including 15 pixels with 40 data points per pixel: instead of displaying 600 points, the envelope displays 48 dots.

Instead of letting the GDI aggregate vertical vectors, we can compute the maximum and minimum values of the data points fitting within each interval and ask the GDI to plot only the envelope of these vectors. This approach adds one intermediate step between reading data and displaying them, i.e., each data point belonging to a given screen interval is compared to local extrema. The envelopes corresponding to each channel displayed are stored within CLineviewButton. Envelopes are built with the help of a scale that maps the file intervals to screen abscissa.

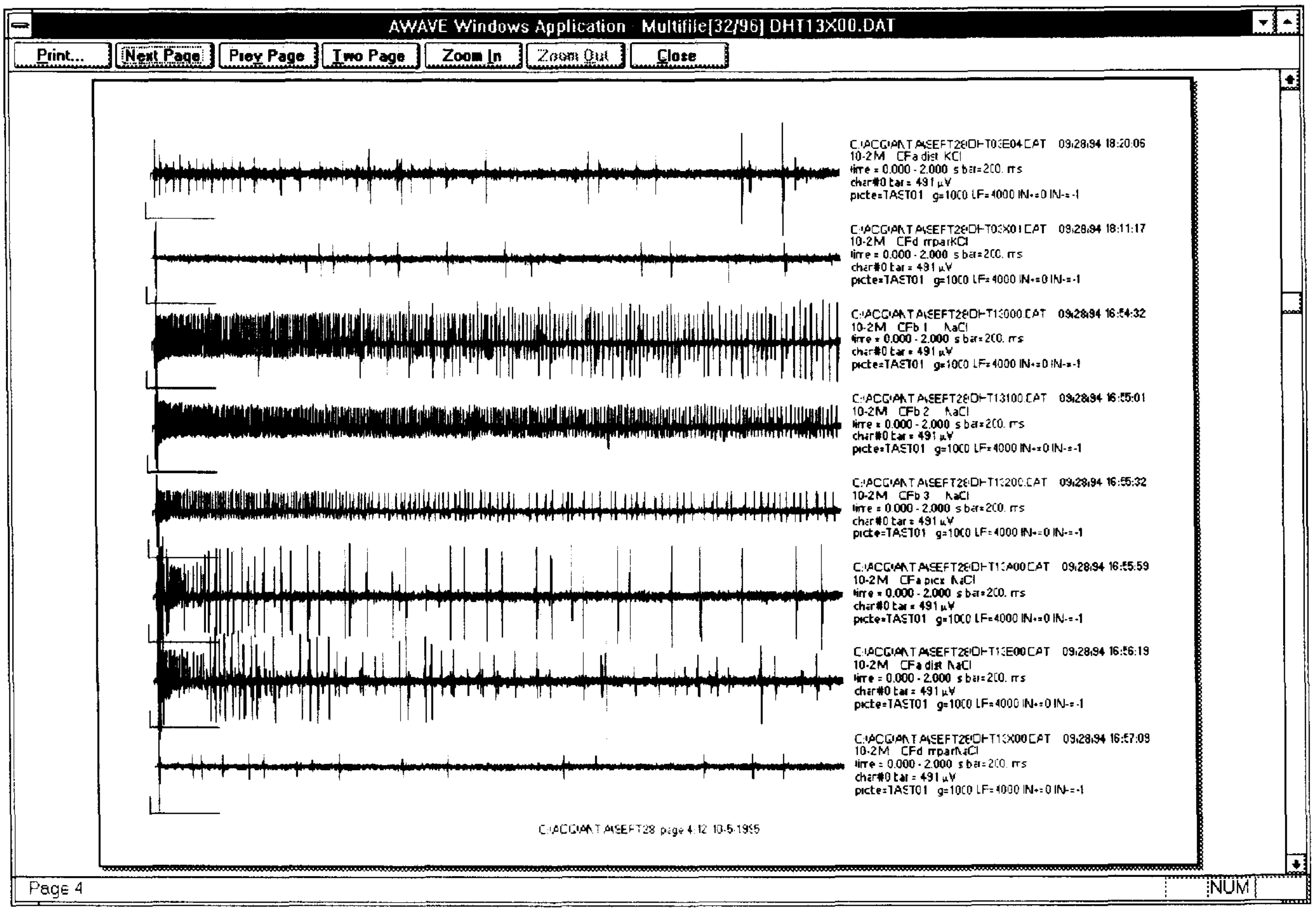
In real-time display applications, this approach has a distinctive advantage over the direct display only if the time necessary to display multiple vectors is higher than computing extrema and displaying the envelope. This should be tested on the target configuration because the display speed depends heavily on a combination of hardware/software factors. On AT-compatible microcomputers, envelope algorithms improve the display speed mainly because memory accesses are much faster than accesses to video memory.

For data analysis, time constraints are less critical but the program is expected to have a good response time. When the initial data representation is built, users often try different amplification factors and vertical offsets in order to examine particular aspects of the biological responses. The envelope approach is definitely superior in these conditions since envelopes do not need to be recalculated whenever the display is refreshed or modified. In addition, envelopes are easier to manage when the mouse cursor is used to select a curve since fewer comparisons are necessary to define a hit.

The envelope approach delivers fast display but possibly slower data accesses. This can be optimized under various ways. For example, computing time can be saved by storing intermediary results that can be reused later. In order to build an envelope spanning a given set of file intervals, we need to estimate how many data points fit into a given interval. This computation does not need to be performed each time but only once, i.e., when the sizes of the window and of the number of data to be displayed are changed. Horizontal scrolling represents a case where significant improvements can be made. The rationale is that if the scroll is small, most data points of the envelope are still valid. Only a small fraction of the envelope needs to be recalculated. This approach does not generate errors if all intervals are equal and if scroll increments are constrained to a multiple of the interval. For unequal intervals, a trade must be chosen between display errors and speed.

4.5. *Multi-file document: class* `CMultiFileDoc`

Typical experiments designed to test the responsiveness of sensory receptors to a set of chemicals yield several hundred data files. The first step in the analysis of our experiments consists in detecting spikes and sorting them on amplitude and form criteria. Groups of files are analyzed with the same parameters, especially when recorded from the same sensilla, the same insect or when they belong to the same set of experiments. Likewise, printing these experiments for visual inspection involves the selec-



```
C:\ACQ\ANT\ASEFT28\DHT03E04.DAT   09/28/94 18:20:06
10-2M   CFa dist KCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT03X01.DAT   09/28/94 18:11:17
10-2M   CFd mparKCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13000.DAT   09/28/94 16:54:32
10-2M   CFb 1    NaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13100.DAT   09/28/94 16:55:01
10-2M   CFb 2    NaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13200.DAT   09/28/94 16:55:32
10-2M   CFb 3    NaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13A00.DAT   09/28/94 16:55:59
10-2M   CFa prox NaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13E00.DAT   09/28/94 16:56:19
10-2M   CFa dist NaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28\DHT13X00.DAT   09/28/94 16:57:09
10-2M   CFd mparNaCl
time = 0.000 - 2.000  s bar=200. ms
chan#0 bar = 491 µV
picte=TAST01   g=1000 LF=4000 IN+=0 IN-=-1

C:\ACQ\ANT\ASEFT28 page 4 12 10-5-1995
```

Fig. 4. Printing preview of a series of data documents. Multifile documents are of great help when the same data analysis is applied on a series of documents. Printing the results of a series of experiments is simpler and allows one to benefit from standard features such as printing preview.

tion of groups of files, sorted either along time, stimulus order or other parameters. Although the MDI standard offers the possibility of opening many files simultaneously within the same program, the user interface is not adequate for easy switching through them. We have derived a multi-file document from standard documents that helps managing groups of files, clarifies the user interface and eases programming the data analysis.

'CMultiFileDoc' was designed to manipulate lists of files. Creating such an object is straightforward under MFC and consists in a direct derivation from the standard CDocument object. Our derived class adds procedures that maintain and manipulate a list of file names. At the present time, only one file from the list is active at a time. Public procedures are provided to browse through the list, add or delete elements or to get information from one element of the list. CMultiFileDoc can be derived to manipulate any document, such as data files or spike files. These new features are made available to CAcqdataDoc simply by deriving CAcqdataDoc from CMultiFileDoc instead of from CDocument.

In order to exploit the possibilities of CMultiFileDoc documents under the MDI framework, specific additions to the user interface should be made. The first step involves selecting multiple files from within the standard dialog box 'FileOpen'. This is done simply by calling the procedure explicitly and changing the value of a flag. Then, the text returned is parsed to extract the path and the individual name of the each member of the list. This minor addition provides the ground of a fully functional multi-file object. User-interface elements should be added to exploit the possibilities of a multi-file document, such as browsing, editing and modifying the list. Browsing through the list is achieved in AWAVE by adding two menu items and two buttons included in the toolbar. Editing the list is currently implemented with a dialog box that can exchange textual data with the clipboard. These additions and other not described here are essentially implemented by trapping Windows messages within CDataView (see example on Fig. 4).

5. Discussion

AWAVE was initially developed to answer two emerging needs in our laboratory. First we wanted to concentrate our programming development efforts on one platform. Secondly a program running under MS-Windows™ benefits from the facilities provided by our office environment, i.e., text processors, spreadsheet, graphics program and network. This approach lead us to reconsider the life cycle of our data, estimate how much time was devoted to data acquisition versus data analysis. As far as our experimental setups are routinely used for insects involving different research programs, this software project adds more flexibility in the use of our experimental setups. Data are now analyzed from any PC connected to the network. This decreases the occupancy of our experimental setups.

The initial development efforts targeted on elaborating an object-oriented program that would serve as a basis to further developments. AWAVE answered basic needs for display and printing of our experimental data. Despite programming in a high-level language and the implementation under MS-Windows™ that is notoriously slow for some graphical applications, we constructed classes that achieve fast and flexible display of large amount of data. The approach outlined here is applicable to other windowing environments and basically involves the manipulation of intermediary representations of the experimental data.

Displaying envelopes rather than raw data represents a new approach to our knowledge. Usually, electrophysiological data are displayed within limited frames, and browsing through the file is allowed by 'paging' the data stream (Dietz et al., 1990). In order to examine closer specific regions of the data, additional tools are sometimes provided to zoom in or out parts of the graph (Dietz et al., 1990). In order to improve the display speed of large data arrays, alternate approaches exist, one of which consisting in plotting only one point under n (n = number of data points/number of horizontal pixels). This way, fewer points are displayed but fast transients may be missed. The envelope approach is superior in that fast events are never missed because the display behaves like a storage oscilloscope. It is also more flexible in that viewing a complete file or part of it, becomes integrated in the same object. Lastly the display time is not dependent on the size of the data chunk.

In order to improve the task of analyzing series of data, we further created multi-file documents that store lists of files belonging to the same group of experiments. This concept is by no mean new since it mimics some properties of a database where each record is represented by an individual file. 'Metafiles' are used by researchers working on a number of small duration recordings (voltage and current clamp experiments). These files reduce the number of files down to a key file indexing data stored in a companion binary file (Turner and Schlieckert, 1990) or to a single data file as in programs like pCLAMP, CED and Strathclyde Software. Metafiles would, however, not suit our needs since file sizes would become too big. We have recently created 'project' documents that incorporate lists of files and are able to open the data documents corresponding to the list. Therefore groups of files can be defined, stored, manipulated and consulted more easily.

From the programming point of view, developing a specific data analysis program benefits greatly from object-oriented programming environments. The existing tools and the program framework are well adapted both to manipulate experimental data and to create a program running under a windowing environment. The objects created and used within AWAVE are of general value and can be adapted to other file header formats and to data encoded under different modes. The encapsulation of Windows user interface elements within MFC framework and the prototyping facilities of the development program ensure that such a program will be able to evolve according to our needs.

Our current development efforts focus on two directions. First, we want to port our existing spike detection analysis and spike sorting routines (Marion-Poll and Tobin, 1991) and develop template sorting (Foster and Handwerker, 1990; Jansen and Ter Maat, 1992) using semi-automated procedures (insect sensilla: Hanson et al., 1986; Mankin et al., 1987; Smith et al., 1990). Most of the routines developed so far will be used as building blocks, with a special mention to the multi-file object and the display button described in this paper. At the present time, we have implemented spike detection and sorting from amplitude criteria. This development involved the derivation of the same type of classes depicted above, namely a spike document and spike views.

Secondly, we want to acquire data under Windows. This work is greatly facilitated by drivers commercially available from most data acquisition card manufacturers. Interacting with a data acquisition card from Data Translation (under the generic DT Open Layer driver) involves sending messages through specific functions and trapping incoming messages to control the data flow. One message for example is emitted by the driver when an acquisition buffer is filled. AWAVE can trap this message, send the corresponding data to a `CLineView` display procedure and return after signalling that this buffer is free for another acquisition. Our current implementation of this outlined approach allows continuous data acquisition and display at 10000–30000 Hz (total throughput) on 1–8 channels with a variable sweep time (using Data Translation DT2821 A/D card and its associated driver). Writing data to a file will proceed likewise but careful testing will remain necessary to allow real-time data storage and display during experiments.

Acknowledgements

I wish to thank Frédéric Gaubert, Olivier Taché (IUT Informatique, Orsay, FRANCE), and Nicolas Chevrier (Maîtrise de Mathématiques Appliquées, Faculté d'Orléans, FRANCE) who contributed to various aspects of this project during the course of their study. Special thanks are due to my colleagues, Michel Renou and Philippe Lucas who supported this projet through their encouragements, helpful discussions and critical reading of this manuscript. Readers belonging to academic institutions and interested in evaluating this software or use it as a basis for further developments may contact the author, preferably via E-mail to marion@versailles.inra.fr.

References

Dietz, M.A., Grant, A.O. and Starmer C.F. (1990) An object oriented user interface for analysis of biological data, Comput. Biomed. Res., 23: 82–96.

Fadda, A., Falsini, B., Neroni, M. and Porciatti, V. (1989) Development of personal computer software for a visual electrophysiology laboratory, Comput. Methods Progr. Biomed., 28: 45–50.

Foster, C. and Handwerker, H.O. (1990) Automatic classification and analysis of microneurographic spike data using a PC/AT, J. Neurosci. Methods, 31: 109–118.

Hanson, F.E., Kogge, S. and Cearly, C. (1986) Computer analysis of chemosensory signals. In: Payne, T.L., Birch, M.C. and Kennedy, C.E.J. (Eds.), Oxford, Mechanisms of Insect Olfaction, pp. 269–278.

Jansen, R.F. and Ter Maat, A. (1992) Automatic waveform classification of extracellular multineuron recordings, J. Neurosci. Methods, 42: 123–132.

Mankin, R.W., Grant, A.J. and Mayer, M.S. (1987) A microcomputer-controlled response measurement and analysis system for insect olfactory receptor neurons, J. Neurosci. Methods, 20: 307–322.

Marion-Poll, F. and Tobin, T.C. (1991) Software filter for detecting spikes superimposed on a fluctuating baseline, J. Neurosci. Methods, 37: 1–6.

Marion-Poll, F. and Tobin, T.C. (1992) Temporal coding of pheromone pulses and trains in *Manduca sexta*, J. Comp. Physiol. A, 171: 505–512.

Renou, M. (1991) Sex pheromone reception in the moth, *Mamestra thalassina*. Characterization and distribution of two types of olfactory hairs, J. Insect Physiol., 37: 617–626.

Renou, M. and Lucas, P. (1994) Sex pheromone reception in *Mamestra brassicae* L. (Lepidoptera): Responses of olfactory receptor neurons to minor components of the pheromone blend, J. Insect Physiol., 40: 75–85.

Skeen, R.S., Van Wie, B.J., Fung S.J. and Barnes, C.D. (1992) Application of compiled BASIC in developing software for collection and analysis of neuronal firing frequency data, J. Neurosci. Methods, 41: 113–121.

Smith, J.J.B., Mitchell, B.K., Rolseth, B.M., Whitehead, A.T. and Albert, P.J. (1990) SAPID tools: microcomputer programs for analysis of multinerve recordings, Chem. Senses, 15: 253–270.

Stromquist, B.R., Pavlides, C. and Zelano, J.A. (1990) On-line acquisition, analysis and presentation of neurophysiological data based on a personal microcomputer system, J. Neurosci. Methods, 35: 215–222.

Stys, P.K. (1991) Neurobase: a general-purpose program for acquisition, storage and digital processing of transient signals using the Apple Macintosh II computer, J. Neurosci. Methods, 37: 47–54.

Turner, D.A. and Schlieckert, M. (1990) Data acquisition and analysis system for intracellular neuronal signals, J. Neurosci. Methods, 35: 241–251.