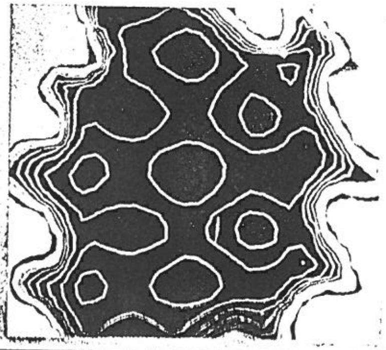
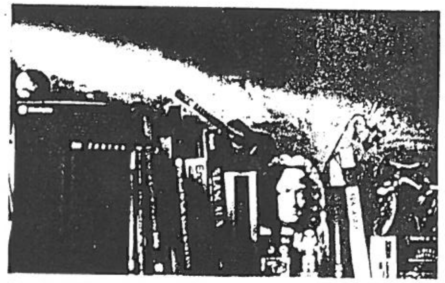
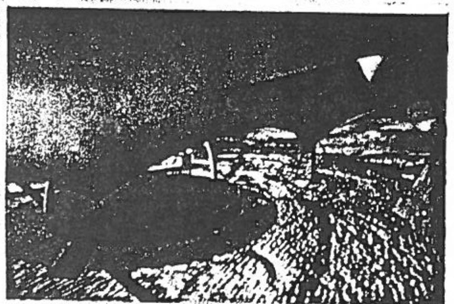
ICOGRAPHICS 93

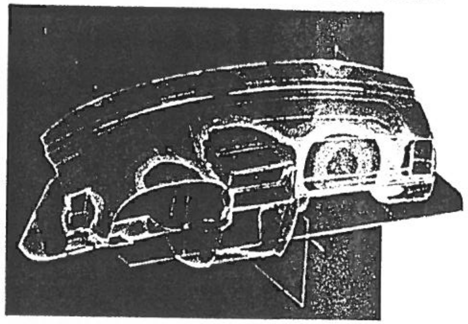
ATTI DEL CONVEGNO

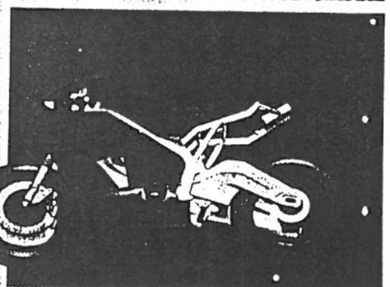
8° convegno internazionale e mostra sulle applicazioni della computer graphics nella produzione, progettazione e gestione

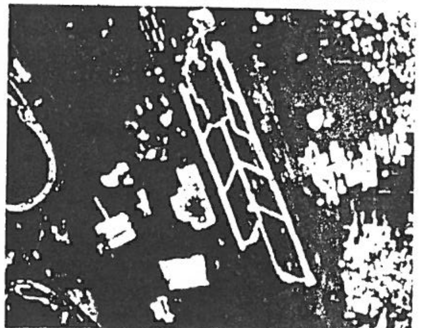












MONDADORI INFORMATICA

icographics

A C++ Application Framework to integrate X Windows and graPHIGS in a coherent object-oriented environment

G. Chiozzi, G. Ghezzi, S. Pantarotto, M. Tucci,
IBM Semea, Scientific and Technical Solution Center
Universita' degli Studi di Milano
e mail: tucci@romesc.vnet.ibm.com

Abstract

Under the AIX operating system, many tools and libraries are widely available for applications in computer graphics. However, they cannot be easily integrated in a coherent programming environment. This paper presents an object oriented approach to the construction of a framework for developers who need to get the best from both X Window and graPHIGS. X Window strength stands in its toolkits, consisting in tools to develop Graphical User Interfaces (GUI) with a standard look and feel; but it lacks in pure graphical functions. On the other hand, graPHIGS is a powerful language for 2 or 3 dimensional graphics programming, but it does not provide high level primitives for the user interface. X Window and graPHIGS are difficult to be mastered and are too much different to be integrated in a single application. We developed an Application Framework, based on a C++ class hierarchy, that provides both advanced graphics capabilities and a powerful environment for building user interfaces. With respect to other tools, this Application Framework not only lets X Windows and graPHIGS communicate, but also creates a new, single, programming environment. The main benefits gained by its use are: a cut off in learning curves and in development times, and a significant reduction of the code size. Thanks to the object oriented approach, the resulting code is easier to read and to maintain; it is also easily extensible and reusable. So far, the library focuses on 2D graphic programming. At the same time, an Interactive User Interface Builder is under development. With this last tool, programmers will be able to create applications using mainly the mouse.

1 Introduction

Many tools and libraries are widely available for applications in computer graphics, under the Unix operating system and its IBM implementation, AIX. They usually can be classified in two distinct classes: on the one hand, X Windows based tools are commonly used to build graphical user interfaces, with a standard look and feel, but they lack in pure graphical functions. On the other hand, advanced graphics is achieved by libraries (e.g. PHIGS and GL) that do not provide high level primitives for the user interface, such as buttons, menus, etc. X Window and graPHIGS are difficult to be mastered and are too much different to be integrated in a single application (a tutorial about this subject can be found in [1]).

Graphical applications usually consist of two main parts: a first one to manage the user interface, and a second one to process commands and data [4]. Most of the code needed to manage the interaction and the graphical display is application independent. Standard procedures and algorithms can be implemented to accomplish this. Basic elements of the user interface, such as buttons, menus, and windows, are "physical" objects that appear on the screen and are "manipulated" interactively. An object oriented approach seems to be the natural and obvious solution.

Starting from a proper set of classes providing skeletons of user interfaces, complete applications can be inherited, by adding new functions and tuning the behavior of existing objects. This is what we call an Application Framework [2]. This paper presents a C++ Application Framework that encapsulates both X-Windows and PHIGS in a coherent programming environment. All the features described here have been implemented and used to develop several applications. The package runs on IBM Risc System/6000 workstations, equipped with AIXwindows and graPHIGS.

2 Design of the Application Framework

This project originated within the IBM Italian Scientific and Technical Solution Center, to answer some specific needs. The basic issue was to provide a framework both to build 2D graphical user interfaces and to serve as a development and debugging tool in an industrial research environment. At the same time, users of the library had no experience in Object Oriented Programming and were not accustomed to the development of user interfaces with X Windows or of advanced graphical applications with PHIGS.

Thus we identified a set of primary goals. Rapid prototyping was the first, to allow a quick development of debugging tools and of application prototypes, with a standard look and feel. All the user interfaces must have a similar style and comply Motif [7] and IBM SAA CUA [6] guidelines. At the same time, a gradual migration to OOP was required, to allow Fortran programmers and C programmers to migrate smoothly toward Object Oriented Programming and reuse their old code. Last, but not least, we wanted to reduce learning times, i.e. reduce the effort required to learn how to develop user interfaces with X Windows and effective graphical routines with PHIGS.

All these issues led to a multi-layers architecture. At the lowest level, two wrappers provide an object-oriented interface to the basic AIXwindows and graPHIGS libraries. In this way, we obtain a coherent environment. The users of both the libraries can easily learn how to develop applications within the new environment; actually the main concepts are preserved in both cases, but their implementation is extremely simplified. A set of base classes implement help, error management facilities, and other features of general use.

Using the first level classes and functions, objects of higher complexity are built. They include tools to implement dialogue panels, to fill in forms or menus easily, to retrieve graphical objects, and to draw something interactively.

At the highest level, skeleton application classes are the core of the Application Framework. In order to develop a new program, the user must choose the most similar sample object and inherit a new class from it.

These layers will be described in greater details in the following sections.

3 The X Windows wrapper

Motif and the Xt Intrinsics are widely used for developing user interfaces in the Unix world. They support an object oriented architecture, through the concept of widget [9, 10], even in a not object-oriented language, like C.

We decided to develop a set of classes embedding widgets into C++ objects. We chose this approach even though object oriented libraries based on X Windows are available. In fact, many of them, such as Interviews [8], are built over Xlib, the lowest X-Windows layer, that does not include widgets. This means that the experienced Motif programmer is set back by the need to approach the new library in a different way. We want to save skills. Moreover, the library developers should re-implement many tasks already accomplished by Motif: this was beyond our scope. Last, Motif look and feel and the use of widgets has become a de facto standard.

We also preferred to build our own library instead of using other available sets of classes built over Motif widgets, such as the Widget Wrapper Library [3]. They do not fit well our needs: in particular, they do not address the problem of the integration with graPHIGS.

3.1 The class tree

The tree of the "widget objects" (figure 1) has its root in the Core_O class, that encapsulates the core widget, and covers all Motif widgets. The hierarchy includes classes obtained in three different ways:

encapsulation of a standard widget : this is done by deriving a class from the object that corresponds to its parent in the widget hierarchy. Macros make such process straightforward. This approach ensures full re-usability of already developed widgets;

classes with extended functions : they are obtained by inheritance from the class that is to be extended. For example, the StringField_O is derived from the Text_O and handles an input field linked to a data variable. The programmer has no need of reading the field to update the variable;

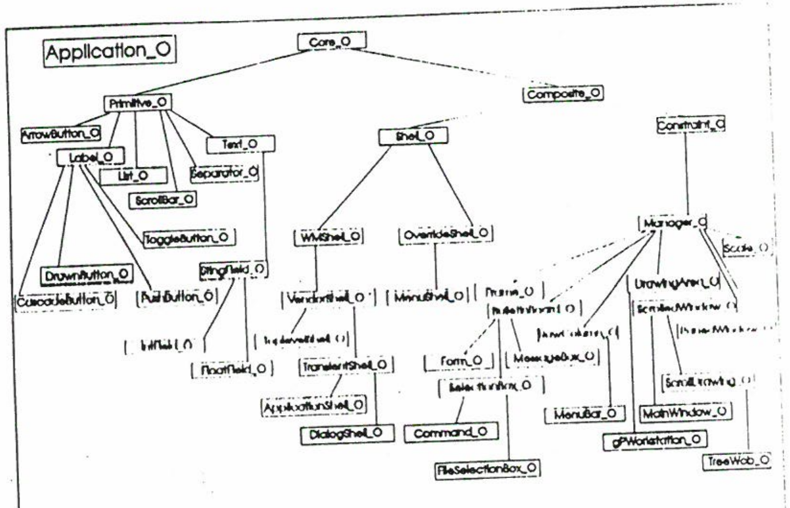


Figure 1: Some classes of the Core_O tree

classes encapsulating several objects : sometimes the design of a new class identifies a main object that manages others. The new class can be derived from that of the main object. The managed objects can be inserted as members. Generally, the main class is derived from the sub tree of Composite_O, the meta class for container widget objects. An example is the Toolbox_O class, a RowColumn_O that manages a number of DrawnButton_O.

3.2 The Application_O object

Every user interface developed with this library requires at least one instance of the class Application_O; this class hides to the programmer X Windows initialization, display handling and a number of boring procedures. Using only the first level classes mentioned above, the traditional Hello World program has the following form:

```
#include <Core_O.h>

main()
{
   Application_O app("app");
   Label_O lbl1("Hello World", app);
   app.Start();
}
```

where the *lbl1* object is a child of *app*, an instance of Application_O class.

The main limit of our library is that it does not overlap all the functions of X lib and X Windows Toolkit. This target is well above our possibilities. In any case, each class is related to the main widget hidden into it. We developed conversion operators that extract the widget in a way transparent to the programmer. As a consequence, whenever the need arises, it is possible to use X Lib and X Windows Toolkit functions easily.

We also provide methods to add new resources to the classes, thus fully supporting the X Windows Resource Database Manager.

4 The graPHIGS wrapper

PHIGS is a powerful graphic language [5]. It virtually answers all the needs of a graphic programmer (except the user interface development). Due to its complexity, however, it takes much time before being used properly and effectively. The concepts related to workstations, viewports, windows, views, and structures, are not readily mastered. The number of routines and parameters is very high, but most applications actually use only a subset of PHIGS routines and functions.

We chose the most suitable subset for our purposes (2 dimensional primitives). We took advantage of the fact that the graPHIGS architecture is suited for an object oriented implementation. A similar approach can be found in [11], where the problem of integration with X Window, however, is not fully addressed.

4.1 The gPWorkstation_O object

The first step towards the integration of graPHIGS and AIX windows has been the development of a widget object to manage a PHIGS workstation: the gPWorkstation_O object.

This class is fully integrated in the Core_O tree and adds a number of specific resources to its ancestor, the DrawingArea_O. It provides a graphic area, where the drawing is performed with graPHIGS. For example, it implements methods for creating and connecting workstations, associating views to them, and dealing with coordinate transformations.

4.2 The other main objects

The encapsulation of the graphic language consists of a set of classes, besides the gPWorkstation_O:

gPView : in PHIGS philosophy, a view is simply a way to display a portion of the world (the window) into a rectangular area of the workstation (the viewport). The methods associated with this class make these concepts intuitive and the implementation of panning, zooming, etc., quite easy;

Struct : the main advantage of PHIGS approach is that the basic graphic element is not the pixel (such as in X Windows), but the structure element (line, polygon, text, etc.). These elements are assembled into complex objects, called structures. They are modifiable at any time, but the programmer does not have to take care of redrawing the screen. The Struct class implements these concepts, and its methods include all the 2 dimensional graphic primitives;

Input : PHIGS offers powerful input techniques. Different logical devices can be associated to the same physical device: they can directly return the input position in the world coordinate system (Locator), or the structure element touched with the pointer (Pick). Classes derived from the Input meta-class implement these input devices. They are fully integrated with the X-Windows input strategy: all X-Windows and PHIGS events are driven in a single queue, managed by the X Server. The application program simply waits for input events from the X Server. Then, it answers through the callback mechanism [9]. This event driven model is the best solution for multi-tasking environments, since an idle application (one with an empty event queue and waiting for input only) can be suspended by the system, thus preventing waste of CPU time.

4.3 The functional approach

All the graphic primitives, beside being methods of the Struct object, appear in the library as functions. Compared to graPHIGS functions, they are much more easily mastered, thanks to a heavy use of useful C++ features, such as "function overloading" and "default parameters".

We chose this approach for a number of reasons: (i) to maintain a "soft" approach to the library for those programmers not familiar with object orientation; (ii) to make the porting of already developed graphic algorithms easier (according to our experience, only a few hours of work are sufficient, in most cases); (iii) to optimize performances. In fact, in applications where a great number of structures are necessary (hundreds or thousands), the *Struct* object can introduce an undesired overhead.

5 High-level tools

On top of these two wrappers, we developed a wide set of classes. They make it easy to implement tasks, which are common to almost every program. Among these classes, Dialogs, Menus and Fill in panels are of fundamental importance: they implement control of the application user dialogue. In this way, the programmer does not need to handle low-level events.

All the above classes are of general use. The library also has a set of classes, explicitly designed for specific applications. In particular, the field of map processing and recognition [12] was chosen as a "test" example of applications for our framework. In this case, the main class is a general purpose map viewer, with navigation functions and the ability of displaying multiple views of the same map. It also provides navigational tools, such as zooming and panning, making the implementation of a complete map editor an easy task.

6 Application classes

The lowest layers of the library provide a much higher-level interface than Motif and PHIGS. However, most applications contain a surprising amount of duplicate code.

Considering only the minimum features common to all our applications, it is true that all programs must:

- declare an instance of Application_O;
- create one or more widget objects defining the user interface;
- create the gPWorkstation_O instance;
- initialize the input devices;
- handle events starting the Application_O by calling its Start() method.

Most programs perform all these tasks with only minor changes. A commonly used approach is to prepare a template with these steps and use a copy of it as the basis of every new program.

A more powerful approach is to capture in a class an application skeleton that performs these steps. Programmers can reuse the common code in a very simple way: an instance of the class must be declared, or a new class must be derived from the original one.

Several advantages stem from this approach. In fact, when a new class is derived and new features are added, it is very easy to go back to the original behavior, because changes are made only to a derived class. In this way, programmers are unlikely to introduce accidental errors in already existing code, because changes are only made to the derived class, and the original code is safely encapsulated in its own class. Enhancements and fixes of the base class are automatically extended to the derived ones.

The power of this approach becomes evident when we apply it to a specific category of complex applications. In such a case, it is easy to identify a wide set of common features that can be encapsulated in an application class.

The library itself provides only a basic set of application classes, but every development group can build its own specific set in a short time. Some of the ready made classes are:

SimpleApplication : a basic skeleton that provides an empty window. It takes care of all the initializations. The programmer must put all the widget objects needed in the window, and then define the actions to be performed in response to events.

MenuApplication : it provides an application with standard menu handling (based on the IBM SAA CUA specifications), error management, help, and message system.

graPhigsApplication : adds to the preceding one a gPWorkstation_O object, with a standard implementation of input handling, hiding all initializations.

Conclusions

In this paper, we have discussed our object oriented approach to the development of applications that require both an effective user interface and advanced 2D graphic capabilities.

We developed a tool that pays attention to the requirements of an industrial research environment: reusability of all the code already developed and fast migration of programmers to the new techniques are of fundamental importance.

The Application Framework is in use by our group since June 1992. Other groups of developers are using it too. These colleagues test our work in a useful way, and their feedbacks have driven a number of extensions and changes to the architecture.

In this way, we have been able to fulfill the most important original goals. In particular, the application classes and the complex objects provide standard look and feel and uniformity between applications. The high level classes also simplify the development of graphic code: this allows rapid prototyping of new applications. Our experience has demonstrated that a C programmer usually needs three days of work, side by side with an expert user of the library, to become productive. After a few weeks, he fully masters the library and uses most of the features of OOP. The possibility of a gradual migration to OOP is of primary importance for this result.

More work can be done to solve the issues identified throughout the paper. In particular, we feel the necessity of a better integration of our library with the programming tools available for the IBM Risc System/6000 workstation. A significant step toward the usability of this library would be an interactive user interface builder, mostly for simple interfaces. This will be the subject of future work.

References

[1] Marchetti M., "Graphigs in ambiente X Window", *Proceedings of I.CO.GRAPHICS*, 1992 (in Italian).

[2] Coutaz J., "The Construction of User Interfaces and the Object Paradigm", *Proceedings of European Conference on OOP*, 1987.

[3] Fekete J.D., "WWL, a Widget Wrapper Library for C++", Laboratoire de Recherce en Informatique, Facult. d'Orsay, Orsay Cedex (France), 1990.

[4] Foley J.D., van Dam A., Feiner S.K., Hughes J.F., "Computer Graphics, principles and practice", Addison Wesley, 1990.

[5] Gaskins T., "PHIGS Programming Manual", O'Reilly & Associates, 1991.

[6] "IBM SAA Common User Access Guide to User Interface Design", IBM Document SC34-4289-00, 1991.

[7] Open Software Foundation, "OSF/Motif Style Guide", Prentice Hall, 1990.

[8] Vlissides J., Linton M., "Applying Object Oriented Design to Structured Graphics", *Proceedings of the USENIX C++ Conference*, Denver, Colorado (USA), 1988.

[9] Young D., "The X Windows System: programming and applications with Xt", Prentice Hall, 1990.

[10] Young D., "Object Oriented Programming with C++ and OSF/Motif", Prentice Hall, 1992.

[11] Wampler S., "Development of a graPHIGS based Object Oriented Graphics System", *graPHIGS User's Group Meeting*, Blacksbourg, Virginia (USA), 1991.

[12] Boatto L., Consorti V., Del Buono M., Di Zenzo S., Eramo V., Esposito A., Melcarne F., Meucci M., Morelli A., Mosciatti M., Scarci S., Tucci M., "An Interpretation System for Land Register Maps", *IEEE Computer*, Vol. 25, No. 27, 1992.