# PhotoPix: an Object-Oriented Framework for Digital Image Processing Systems

Alisson Augusto Souza Sol
Arnaldo de Albuquerque Araújo

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702
30.161-970 Belo Horizonte, MG, Brazil
{alisson | arnaldo}@dcc.ufmg.br

**Abstract.** This work describes how the use of object-oriented technology can help to separate the implementation of algorithms in digital image processing systems from the coding of "non-essential" functionality. The goal was the design and implementation of a system to allow researchers of digital image processing to keep their attention in the issues related to the specific algorithm they are implementing, without worrying about image reading/writing, image displaying/printing, user interface, etc. It describes a framework for digital image processing systems, named PhotoPix, which was designed using object-oriented methodology and coded in the C++ language, using the application programming interface of the Microsoft® Windows™ environment.

## 1 Introduction

The continuous evolution in microelectronics technology has made possible the production of microcomputers with greater computational power than most past decade mainframes. With the available personal computers, users expect to produce multimedia presentations instead of just plain text documents. So, there is an increasing interest in digital image processing (DIP) systems.

The purpose of DIP systems is the enhancement of desired attributes of digital images and the automation of procedures for information extraction, such as segmentation and pattern recognition [1,2].

Currently, two big problems pervade most implementations of DIP algorithms:

- the numerous formats for files storing digital images [3,4];
- the incompatible hardware platforms and operating environments available nowadays, making difficult the portability of user interface code.

Being almost impossible to code a system with support for every known file format, hardware platform or operating environment, developers must concentrate efforts on a reduced number of choices, based on available technology and resources. However, those choices can make the system useless after a few months, as technical and commercial issues have rendered unused several once popular hardware platforms, operating environments and file formats.

## 2 System Overview

The goal of the PhotoPix system is to provide a theoretical and practical basis for object-oriented DIP systems. The system should be portable not only to the several hardware platforms and operating environments of present time, but also to the future ones, making real the concept of reusable software [5,6].

Many efforts are usually duplicated when programmers implement DIP algorithms, because most available source code can not be reused. One of the main reasons for that is the difficulty to separate the algorithms code from user-interface code, file manipulation code or other code only needed due to restrictions of hardware technology. In the PhotoPix system, the code implementing DIP algorithms is separated from the code that deals with "non-essential" functionality.

Nowadays, the available hardware is closely approaching the perfect technology concept of Essential Analysis [7]. Therefore, for a system to be portable over time, it must not depend on any particular hardware characteristic -- or it must have minimum dependencies and they must be clearly isolated. Several available APIs (application programming interfaces) already make possible the coding of DIP systems without any references to the hardware. Among them, it was chosen the Microsoft Windows API [8,9] for the first implementation of the PhotoPix system.

Object-oriented programming is currently the best available option to code systems where one needs to encapsulate details and separate functionality in several modules. To code the PhotoPix system, it was chosen the most popular object-oriented programming language: the C++ language [10]. As there is not that same consensus about methodologies for object-oriented analysis and design, several techniques were used to specify and design the PhotoPix system, the main ones being Essential Analysis [5] and the Booch methodology [11].

A widespread solution for the problem of dealing with many different image file formats is the adoption of a unique internal image data representation -- encapsulated from algorithms by access functions of the image abstract data type. Converters between the internal format and some of the several popular image file formats allow the system to exchange data with other systems.

A major problem faced when choosing a data structure for image representation is the number of bits used to represent the quantized spectral information about each pixel. Most systems just adopt a standard spectral resolution, usually 8 or 24 bits/pixel. Those systems will not be able to read/write images from/to several popular file formats. Another common solution is to allow any -- or, at least, many -- spectral resolutions and color systems. However, the usual penalty in this approach is the obligation for the programmer to code many versions of the same DIP algorithm.

To avoid programmers having to code several versions of the same DIP algorithm or to use a generic and inefficient one, the PhotoPix system allows implementation of algorithms that might be executed only on specific spectral resolutions. The algorithm is queried about its ability to process the input images and, if it can not generate a reasonable result, it is not allowed to execute. A possible penalty in this approach is the need for the end-user to convert an image to another spectral resolution before being able to use a particular algorithm.

Several algorithms to convert the spectral information between images with different resolutions were studied. The most difficult issue is the reduction of the number of bits used to store spectral information for each pixel. To solve that problem, it is usual to map the original color space to a reduced one created by uniform quantization of the full spectrum, the popularity algorithm or the median cut algorithm [12]. Other considered issues were the rendering of images using a gray scale palette and local error diffusion [13,14].

# 3 System Specification

Essential systems analysis [7] was the main methodology used to specify the PhotoPix system. In that technique, the analyst must suppose that the system under specification will be executed by a machine built using "perfect technology" (perfect and infinite memory and an ideal processor with infinite execution speed). So, it is not necessary to list usual requirements that only deal with technological restrictions, as "read file into memory" (something not needed if there were "perfect memory") or "schedule process execution" (something that is usually needed just because a process is computationally so expensive that it can not be executed on-line with currently available hardware).

For a specification done with the assumption of execution under perfect technology, the inevitable requirements for a system lead to what is known as "system essence", formed by the union of the essential activities that the system must execute and the data that must be available for the system to execute them. However, it is important to recognize system borders, so that the analyst does not ignore usual limitations of humans and other systems with which the new one must communicate.

## 3.1 PhotoPix Essence

The essential memory of the PhotoPix system is a repository of images. Its fundamental essential activity is the execution of algorithms, transforming the images in the repository or generating new ones.

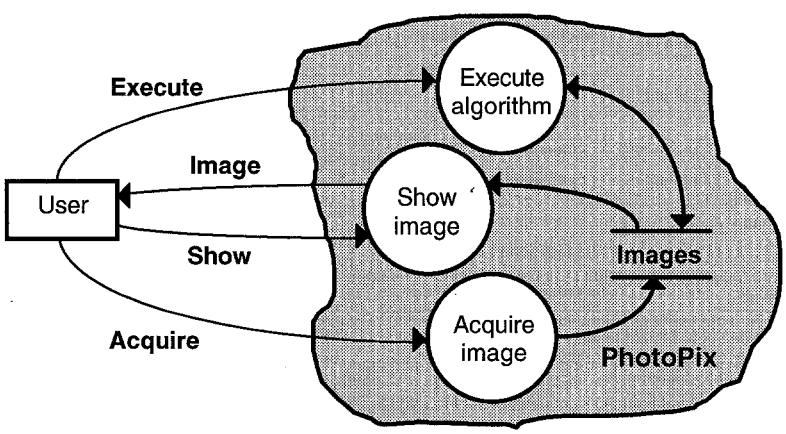


Figure 1 Essential model of the PhotoPix system

The "high-level" essential model of the PhotoPix system, shown in Figure 1, also presents the secondary essential activities of acquiring images from the external world to feed the image repository, and displaying those images in some output device. All those activities will be executed at user's discretion.

# 4 Design and Codification

The essential model of a software system can be implemented in several different ways. However, most procedural implementations lead to a result that barely resembles the specification. Additional potential problems for digital image processing systems are their high requirements for the hardware platform. Most programmers do not resist the temptation of getting a better performance by using low-level calls to functions available in just a particular video board or graphics accelerator. That code is rarely portable.

A very useful abstraction available today to help in the development of portable code is that of an API (application programming interface). An API can be seen as a documented set of function calls, their implementation for some hardware systems and the specification of a mechanism to use that functionality from code written in some computer language. Through the API, a "virtual machine" is made available to the programmer, helping in the preservation of the essential model in the final code.

After some research and evaluation of several alternatives, the Microsoft Windows API [8,9] was chosen to code the PhotoPix system. That graphical API is currently available to several different hardware platforms, being directly implemented or through emulation.

The use of an API can be a burden to a software system, specially in hardware systems where the API is emulated. Even in that case, the programmer can benefit from the similarity among the several available graphical APIs and make use of the most suitable to a particular platform. However, another solution is the use of class libraries, which are even higher level abstractions available in some object-oriented languages. The PhotoPix system makes use of the MFC (Microsoft Foundation Classes) library [7].

## 4.1 Identifying Classes

The design of an object-oriented system starts with the identification of abstract data types and their functionality. That can be done by using scenarios, which are supposed snapshots of the final software system when in use. In those snapshots, several objects (class instances) appear collaborating to achieve the system functionality, making possible to identify the semantics of classes and objects.

From the essential model presented in Figure 1, it is easily seen that there might be an image class in the PhotoPix system. That class would offer basic functionality for the execution of the digital image processing algorithms. An abstraction that is harder to be seen by people used to the development of procedural systems is that of an algorithm class. Instances of that class would be called to execute their functionality in some input images, modifying them or generating new ones. The

basic functionality would be implemented in an abstract base class. By using inheritance, a new class would be defined for each particular DIP algorithm.

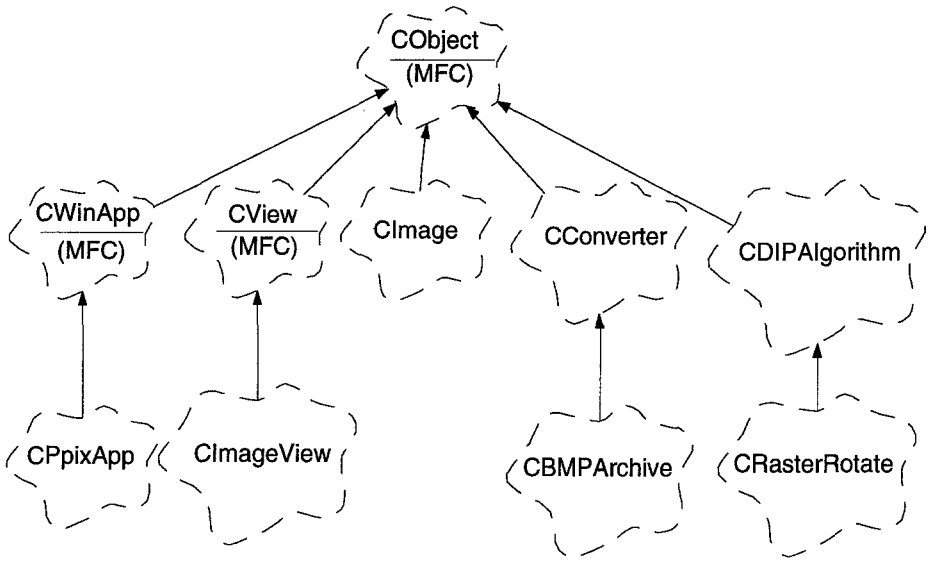


Figure 2 PhotoPix class hierarchy

The class hierarchy used for the PhotoPix system is shown in Figure 2, using the notation of [11] (only inheritance relationship is shown and some levels have been omitted to simplify the picture). The image and algorithm classes are the fundamental ones for the PhotoPix system. However, there is the need for classes that define how messages would flow between the user and the internal objects. That functionality is provided by the document-view architecture of the MFC library [9].

One object of the *CPpixApp* class is used to control the execution of the PhotoPix system. Objects of the *CImageView* class act like lens through which one can view the objects of class *CImage* in the system repository. The *CDIPAlgorithm* class is the basis for derivation of classes that would implement DIP algorithms.

The image repository would ideally be maintained in memory. However, that is not possible with current technology. That leads to the need for the *CConverter* class, which is the basis for classes that make the conversion between the internal representation of image data and image file formats, providing persistence for the image objects.

## 5 Conclusions

It was presented a simple structure for digital image processing systems, which helps to achieve reusability and efficient use of resources when doing research about DIP algorithms. Using object-oriented technology, it was designed a framework for DIP systems, which had a successful implementation for the Microsoft Windows API, using the C++ language and the MFC library.

Preliminary versions of the PhotoPix system were used in courses on digital image processing offered at the Computer Science Department of the UFMG. After a brief presentation of the interface and mechanisms of the image and algorithm classes, most students with basic understanding of the C language could add at least one DIP algorithm to the system in less than a week. This result indicates that a great level of abstraction was achieved.

The system is now being extended to enable the end-user to create new algorithms by composition of the basic ones. Also, the kernel subsystem is being extended to distribute processing among instances of the system running on different machines. In another future extension, the system will be made able to execute DIP algorithms over images in multimedia sequences.

## 6 Acknowledgments

The authors would like to thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (grants 400.190/90-7, 500.908/91-5 and 130.990/90-6) and the Fundação de Amparo à Pesquisa do Estado de Minas Gerais - FAPEMIG (grant TEC 1113-90) for the financial support of this work.

## 7 References

1. Jain, A.K., *Fundamentals of Digital Image Processing*, Prentice-Hall, Inc., 1989
2. Pratt, W., *Digital Image Processing*, John Wiley & Sons, 1978
3. Kay, D.C. & Levine, J.R., *Graphics File Formats*, Windcrest Books, 1992
4. Murray, J. & van Ryper, W., *Encyclopedia of Graphics File Formats*, O'Reilly & Associates, 1994
5. Cox, B.J., *Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Inc., 1986
6. Martin, J., *Principles of Object-Oriented Analysis and Design*, Prentice-Hall, Inc., 1993
7. McMenamin, S.M. & Palmer, J.F., *Essential Systems Analysis*, Prentice-Hall, Inc., 1984
8. Petzold, C., *Programming Windows 3.1*, Microsoft Press, 1992
9. Microsoft Co., *Microsoft Visual C++ compiler manuals*, Microsoft Co., 1995
10. Ellis, M.A. & Stroustrup, B., *The annotated C++ reference manual*, Addison-Wesley, 1990
11. Booch, G., *Object-Oriented Analysis and Design - with Applications, 2nd. Ed.*, The Benjamin/Cummings Publishing Company, Inc., 1994
12. Heckbert, P., Color image quantization for frame buffer display, *Computer Graphics*, Vol. 16, No. 3, Jul. 1982
13. Ulichney, R., *Digital Halftoning*, MIT Press, 1988
14. Sol, A.A.S., *PhotoPix: an object-oriented framework for digital image processing systems*, Master thesis, DCC/UFMG, Belo Horizonte, MG, Brazil, 1993 (In portuguese)