MIPL : An Image Processing Library for Medical Applications

C. Bouras[1,2], *T. Georgantas*[3], *K. Goutis*[4], *V. Kapoulas*[1] *and P. Spirakis*[1,2,5]

[1] *Department of Computer Science and Engineering, University of Patras, Greece.*
[2] *Computer Technology Institute, Greece.*
[3] *National Technical University of Athens, Greece.*
[4] *Department of Electrical Engineering, University of Patras, Greece.*
[5] *Courant Institute of Mathematics, USA.*

1 Introduction

The display and manipulation of medical images is becoming a very important part of PACS (Picture Archiving and Communication Systems) and HIS (Hospital Information Systems) [6]. The access and manipulation of medical images in digital form require different functionalities. The most common are visualization, image file transfer, storage in a database etc. These functionalities are most useful when they are unified. Different unification models exist but the overall concept is very simple. There must be the ability to display, enchance and process medical images in a multi–purpose workstation, to intergrate different image and medical modalities in a consistent way and to communicate and exchange information between different systems, possibly located at different places.

Towards this direction the design and the implementation of a Unified Medical Workstation (UMWS) was decided, as part of the TELEMED project. A very important low–level part of the UMWS is MIPL. MIPL (standing for Medical Image Processing Library) is an image processing library for medical images. MIPL is essential for supporting all kinds of image manipulation requiried by the UMWS. MIPL is part of the Display Tools (DSP), which, in turn, are part of the Unified Presentation Tools (UPT). These are used to implement the Unified Medical Workstation Tools (UMWST), which are necessary to support any application written for the Unified Medical Workstation (UMWS). Although MIPL will be used for medical image processing, it is designed to be used as a general–purpose image processing library also.

Other programs for manipulation of medical images are EXPLORER [2], OSIRIS [3] and LiteBox [4].

2 General overview

2.1 Features

MIPL [5] is designed to handle images that are produced by different modalities. These images are very different with respect to some parameters that characterize

certain attributes of them. Examples of such parameters are image size, image depth, type of image data etc.

MIPL is able to handle images of very different sizes. In fact there is no restriction in the size of an image that MIPL can handle. The only limit that applies to the size of an image is the memory available in the target workstation that executes some function of MIPL.

Apart from different sizes, images can have different depth. By depth we mean the number of bits needed to represent the gray scale level of one pixel in the image. Bigger depths means that there are more gray scale level for the visualization of the image and thus a physician can have better understanding of this image. MIPL can handle images with any depth, as long as this is less or equal to the number of bits used to represent the `long int` type in the target workstation.

Additionaly, MIPL supports different types of image data. The supported types are byte, word and double word either signed or unsigned and single or double precision float. That is, MIPL will accept and manipulate any image that has its data represented as a sequence of numbers of any of the above types.

MIPL can handle images of higher dimensions (3–D and 4–D). 3–D images can be though of as either true representations of 3–D objects (a tomography) or as time sequences of a 2–D objects. 4–D images can be though of as time sequences of 3–D objects. MIPL supports any image processing operation that can be defined in these higher dimensions (there is a certain difficulty in defining or extending the meaning of some operations in more than two dimensions).

At last, MIPL can handle parts of images, the so–called ROIs (Regions Of Interest). ROI handling is very useful when only a small part of the image is of interest. In such a case image processing can be done in that area only, saving a considerable amount of time.

2.2 Advantages

MIPL is implemented in the ANSI–C programming language. It does not depent on any specific attribute of the computer and it does not require any special–purpose hardware. So it is hardware independent and can be ported to any machine. That is very important because MIPL will be used for medical applications and currently the computers used for medical applications are of very different type even within the same hospital.

MIPL has an object–oriented design. Apart from the ease of implementation that this design gave us, it also hides the details of the implementation from the user of the library. This way one does not have any access to the internals of the MIPL, cannot alter any attribute and cannot use “quick and dirty” programming. Thus, a good numbers of serious errors may be avoided.

The above is not the only prevention MIPL takes against errors. It, also, incorporates an error checking mechanism. MIPL checks all parameters against

meaningless, erroneous, or impossible values. When a parameter is of a higher level type all the attributes of this parameter are check to make sure that their values are compatible with each other. MIPL, also, checks for errors that might occur during the execution of one of its functions. When an error is encountered MIPL identifies it and informs the higher level (the application using the library) about the existence and the type of the error. This is done by returning an error code and setting an internal variable to an appropriate value.

2.3 Disadvantages

Image processing is a very time–consuming task and usually it needs the assistance of special hardware. On the other hand MIPL is harware independent and cannot rely on any special hardware to speed operations up. Thus MIPL is not very fast (Although it seems to be fast enough when executing in a workstation, this may not be the case when used in a personnal computer). In order MIPL to be as fast as possible, the code of MIPL is optimized to achieve full speed.

Also, MIPL seems to be large in size. This is a consequence of the ability of MIPL to handle images with very different types of image data and the speed optimazation. Simiral portions of code had to be repeated for every one of the supported data types. This is not much of a problem if one considers the amount of memory that a typical workstation has and the fact that a typical medical application may use only a part of MIPL and not the whole library.

A good way to imrove the speed and reduce the size of the library is to use special–purpose hardware. The library will not be any more harware independent but may still be portable if one chooses to use hardware that is widely available for the most commonly used personnal computers and workstations (UNIX machines).

3 Structure of MIPL

3.1 Overall Design

The internal design of MIPL is object–oriented. This means that everything used or manipulated by MIPL must be an object. This design greatly simplifies the implementation of the library, reduces the errors during implementation and makes debugging much faster and easier. This is very important, if one considers what a task is to implement a fully–featured image processing library.

Another great benefit of the object–oriented design is data hiding. All implementation details are hidden from the user of MIPL (the application in the higher level) and the basic object manipulation is obtained through the use of some primitive functions (methods) of MIPL. These are not image processing functions but they are provided by MIPL in order for anyone to be able to handle the objects supported by MIPL.

The effect of the particular design is that the user of MIPL (application programmer) is somehow "forced" to use good programming techniques. This prevents him from making several minor or major errors and, thus, speeding up his job and saving him a greatdeal of effort.

The objects supported by the library are image, histogram, look–up table and region of interest. Each object is manipulated by several functions provided by the library.

The implementation of the library had to be done in the ANSI–C programming language, which is not an object–oriented one. The implementors of the library used several techniques to avoid the limitations and achieve a more object–oriented implementation.

The primary data types used in conjuction with MIPL are :

- Rc (return code)
- Image
- Histogram
- Lut (look–up table)
- Roi (region of interest) — In two dimensions only.

They are defined by common ANCI–C definitions

In order to achieve data hiding the user of MIPL is not allowed to see these definitions. The definitions that he sees look like

```
typedef void *Image;

typedef void *Historgram;

typedef void *Lut;
```

which means that the above types are pointers to something but no clue is given to what this "something" might be.

This way he cannot see the details of the impementation (A basic idea behind the object–oriented design).

The functions provided by MIPL are covering all the topics in the medical image processing field and they are, generally, not depending on one another. This was a great convinience for the implementors of MIPL since every new image processing function that was added to the library could be tested and debugged separately.

The independence of the library functions, also, has another advantage. When one wishes to use only a limited number of the library functions, only these functions will be linked and so the resulting program will not be large in size.

The functions of MIPL are organized in three levels. This decision was made by the designers of MIPL because for the application of some operators a certain preprocessing of the images must take place. So there exist functions for both the preprocessing and the final processing.

In the first or low level belong the functions that perform the preprocessing. In the second or medium level belong the functions that combine the results of the preprocessing to obtain the fully processed image. In the third or high level belong the functions that are responsible for calling the low level functions to do the preprocessing and the medium level functions to do the combination of the intermediate results.

All the functions in every of the three levels are accessible from the potential user. This way one can choose to, only, preprocess an image (if one is interest in the intermediate results) or to combine previously preprocessed images (without preprocessing them again).

3.2 Modules of MIPL

Image processing operators are categorized according to their function and domain of application [1]. This categorization is heavily used in the literature. The designers of MIPL used this fact and decided to organize MIPL in a modular way, according to the previously mentioned categorazation.

MIPL includes modules for the following topics :

- Mathematical operations between a number and an image.
- Mathematical operations between two images.
- Logical operations between two images.
- Image transformations (FFT, inverse FFT etc.).
- Spatial filter operations.
- Band-pass and band-reject filtering (frequency domain).
- Histogram operations.
- Contrast and image enchancement.
- Edge detection.
- Zooming and replication.
- Image type conversion.

The above modules does not include the basic object manipulation functions. (These functions are not image processing functions but they are used to support primitive object manipulations)

3.3 Error handling

MIPL has built in an error checking mechanism. This mechanism is a very important part of MIPL. The designers of MIPL paid a lot of attention when designing this mechanism and made it compact and robust. The error checking mechanism checks against errors made by the user of MIPL (probably an application), faulty parameters and erroneous conditions during the application of an image processing operator.

Before going on and describing the error checking mechanism, we must make clear what is an error for MIPL. For MIPL there are two kinds of abnormal situations. The first one is when MIPL cannot continue the application of a function, probably due to some invalid parameters and subsequently cannot produce a result. The other abnormal situation is when MIPL has finished its processing, produced a result and detected some problem, but it is possible for the application using the MIPL to ignore the error and continue

An example of such a situation is the overflow problem. This occurs when there is not enough memory allocated for the image resulting from an operation of MIPL. In this case MIPL discards some bits of some pixels producing an image with some pixels altered and reports the abnormal situation. From now on the application can either continue, ignoring the fact of some pixels being damaged, or stop and report the error to the user.

MIPL can detect both of the above situations and will report them. When a function of MIPL terminates normally it returns a code of `MIPL_OK`. This means that everything went fine. If an error is detected by the error checking mechanism then MIPL will return either `MIPL_ERROR` or `MIPL_WARNING`. The first code means that the error was fatal for MIPL and no output is produced. The second code means that MIPL did its best and produced some output but with possible loss of information. The application using the MIPL has now the responsibility of continuing.

When MIPL reports an erroneous situation, it is possible for the application to figure out what went wrong by calling the function `mipl_get_error()`. This function will return a code that shows what exactly what the problem was. After that the application can take some actions to correct the situation and resume execution.

4 Conclusions and further work

In this work we have described MIPL, one medical image processing library. MIPL is the first approach in the area of object–oriented programming for medical image processing. The overall design of MIPL is sophisticated and up to now the library is hardware independent.

The next step will be to use special hardware to achieve more speed and efficiency. In order not to loose the portability we shall choose hardware that

is widely available for every personnal computer and workstation. Another step will be to use parallel hardware image processors. This way when one wishes more speed he must only add some more hardware modules.

Finally we have start the development of a user frienty interface that uses MIPL under X–Windows.

References

[1] "Digital Image Processing", R. C. Gonzales and P. Wintz, Addison–Wesley Publishing Company, 1977.

[2] "The EXPLORER", User Manual, O. Ratib, UCLA School of Medicine, Dept. of Radiological Sciences, Los Angeles, 1989

[3] "OSIRIS User Interface and Functionalities", O. Ratib and Yves Ligier, Hopital Cantonal Universitaire de Geneve, Imagerie Reg. 90025, June 1990.

[4] "Litebox", Siemens Medical Systems, Data Sheet, Order No. A91004–M2020–G010–01–4A00.

[5] "MIPL (Version 1.1)", C. Bouras, T. Georgantas, K. Goutis, V. Kapoulas and P. Spirakis, Technical Report 91.04.12 CTI, Patras.

[6] "A General PACS–RIS Interface", Lecture Notes in Medical Informatics, Edited by P. L. Reichertz and D. A. B. Lindberg, Springer–Verlag, 1988.