A DIGITAL IMAGE PROCESSING TEACHING AID

Torsten Paul Nelson

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Caixa Postal 702, CEP 30161-970, Belo Horizonte, MG, Brasil
Fax: +55 (31) 448-5858
e-mail: torsten@dcc.ufmg.br

Abstract

A tool developed for use in an implementation-oriented digital image processing course is described here. It seeks to benefit both students and teachers, the former with an easy to use environment which provides means to open, visualize and manipulate images, and allows modules to be added very easily, and the latter with an open tool which can be easily adapted to fit the requirements of one’s digital image processing course, allowing the teacher simple ways to verify the students’ progress and to provide sample commented source code to be studied or modified.

keywords: digital image processing, computerized teaching aids, image processing tools, image enhancement, edge detection.
1. Introduction

This paper presents a digital image processing (DIP) [Gon87] [Ros82] tool developed as a teaching aid for use in undergraduate- or graduate-level DIP courses. In these courses, the student must deal with a large amount of information, such as algorithms for spatial and frequency domain filtering, for edge detection, image enhancement, segmentation and classification, as well as different formats for image storage [Nib86]. In order to implement any of the techniques learned, however, the student must first invest a significant amount of time developing low-level program modules to input images, store them in memory and display them in a graphic screen. The tool described here is in itself a powerful image processing program with numerous features, but its development was geared towards use in an academic environment. It provides students with a complete DIP environment to which they can effortlessly add their own routines and immediately see results and statistics relative to their execution. The teacher, also, can easily configure the tool to conform to the course’s structure, giving the students access to commented source code for the many routines already implemented, if so desired.

There are several goals behind this work. First, we seek to free the student from image display tasks, thereby allowing him more time to work on image processing algorithms. Second, we wish to provide a flexible environment in which the teacher is completely free to use the tool in the manner that better fits his course. Finally, we want an open tool which is powerful in its own right, and which does not fall behind as the field of digital image processing advances.

2. Tool Description

The tool was developed using the X-Window system [Sch86], and therefore requires an X-Window compatible display. A color monitor is also required. Color depth may be either 8 or 24 bits, and the system can run in either color or grayscale mode. Implemented in the C language on a Sun Sparcstation, it should be easily ported to other types of computers.

Upon startup, five windows are displayed on the screen. The first two are image buffers where images may be loaded. The third is a result buffer, inside which are displayed the results of the execution of the various algorithms upon one or both image buffers. The fourth window contains several command buttons, some of which bring forth pop-up menus. The contents of the command window are variable and depend on how the tool is installed. Finally, the last window shows text information such as explanations and timing measurements for the image processing functions. Figure 1 shows the buffer windows containing small grayscale images and a sample command window.
Figure 1: Image buffers and a command window

Only a few of the command buttons are mandatorily present in every command menu. They are: change active buffer, button, histogram, swap buffers, load file and save file. The change active buffer button is used to select which of the three buffers will be used as input in an operation. It should be noted that the result buffer can also be used as input. The output of every image processing operation is always displayed on the result buffer window. The button command selects which operation will be performed when the user clicks the mouse button over one of the image buffers. Functions already implemented are line profile, column profile and zoom, though others may be added. The histogram button displays a new window containing the histogram relative to the active buffer, as well as other statistical data. The swap buffers button swaps the image in the active buffer with the image in the result buffer. Finally, the load file and save file button open pop-up menus which contain the various file formats which the program can read from and write to. Selecting one of these formats brings forth a dialog box requesting the name of the file to load or save.
The remaining command buttons are used to call image processing functions. They can either display pop-up menus when activated which contain several callable functions or simply call the functions straight off. Command buttons and menu items are configured by the user or users as explained in the following section.

Many program modules have already been implemented, such as I/O for GIF, TIFF, pbm and raw image files; eight types of smoothing filters; five edge detection filters; fast fourier transforms; bayesian classification; two types of segmentation; histogram equalization, and ten types of special effects. The commented source code for all these functions may be given to the student for study, if so desired.

3. Adding menus and menu items

Upon startup, the system examines its home directory looking for C source files and subdirectories. If the C files haven’t yet been compiled, then the system does so automatically. For every C file it finds, a button will be created in the command window with that file’s name. When that button is activated, the system calls the first function contained in the C source file. For every subdirectory it finds, a button will be created which calls a pop-up menu. The items in this pop-up menu will be the C source files that exist inside the subdirectory, and selecting one of these items causes the system to call the appropriate function.

Figure 2 shows a command window and the files existent in the system’s directory. The contents of pop-up menu “Filters” are also shown.

![Command window](image)

**Contents of DIPtool directory:**

- button/
- filters/
- invert.c
- load_file/
- save_file/
- xdiptool*

**Contents of filters directory:**

- 3x3average.c
- sigma.c

*Figure 2: Command window generated by the system for a sample directory structure*
4. Implementing new modules

There are three types of program functions that can be created by users: button, I/O and image processing functions. Button-type functions are those that may be called when the user presses the mouse button over a pixel in one of the images. I/O functions are used to load and save image files of some type. Image processing functions are those that perform some sort of operation on one or two images and generates a result image. All functions must be implemented using the C language and follow a few simple guidelines:

- The program module must include the header file "xdiptool.h".

- The function to be called by the system must have the same name as the C program file, i.e. if the file is called invert.c, then the called function must be named invert().

- Any other functions comprising the module should have the called function's name as a prefix to avoid naming conflicts with other modules, i.e. for the example above, an auxiliary function in the same module might be named invert_otherfunction().

- The called function will receive certain parameters from the system, depending on its type:

  - **Button-type functions**: function_name (source_image, source_columns, source_lines, result_image, depth, button_x, button_y)

    - source_image is a pointer to the active buffer's image data.
    - source_columns and source_lines give the active buffer's image dimensions.
    - result_image is a pointer to the result image buffer. The user need not allocate memory for the result image; however, it should have the same dimensions as the source image.
    - depth is the number of bits per pixel in the source image, and it may be 8 (for grayscale images) or 24 (for color images).
    - button_x and button_y are the x and y coordinates of the location where the mouse button was pressed, relative to the image's top-left corner.

  - **I/O functions**: function_name (source_image, source_columns, source_lines, depth)

    - source_image points to the image's location. If the function is loading an image, then the user must allocate sufficient memory to hold it and return the pointer in this parameter.
    - source_columns and source_lines are the image's dimensions. Again, if the function is loading a file, then its dimensions are returned here.
    - depth is the number of bits per pixel.
- Image processing functions: function_name (active_buffer_image, abi_columns, abi_lines, other_buffer_image, obi_columns, obi_lines, result_image, depth)

- The active_buffer_image, abi_columns and abi_lines parameters are relative to the active buffer, while the other_buffer_ and obi_ parameters are relative to an image buffer which is not active (used for two-image operations). The other parameters, result_image and depth, are as detailed under button-type functions.

* If the function requires user input, it should use the User_Dialog (prompt, user_string) function, which brings up a simple dialog box containing a prompt message, and returns the string entered by the user. A sample dialog box is shown in figure 3.

![Input dialog box](image)

**Figure 3: A sample dialog box for user input**

In addition to the function itself, the programmer may create a text file containing useful information about the module, such as a description of the algorithm or implementation details. This text file should be placed in the same directory as the program module, and have the same name with a .txt instead of a .c suffix. Its contents are displayed in the system's information window when the corresponding function is selected.

5. Using the tool in a digital image processing course

5.1 Under student control

One way in which this system can be used in a DIP course is to hand each student a copy of the program and allow him/her to create his own modules according to the course’s schedule. Milestones may be set in which the student must turn in a specific set of modules. Sample source code for some functions can be handed to the student for study or modification. At the end of the course, each student should have a full image-processing system implemented which can be used as a means for evaluating the student’s progress.
5.2. Under teacher control

The teacher may choose to have a single copy of the system which students have permission to access. This might be a fully shared environment in which the class as a whole collaborates upon the same system, or a partially shared scheme in which each student can access a specific set of directories, but only the professor or assistant can use the system as a whole. For example, each student might have a directory in which he implements his modules. The teacher can then access each student’s modules sequentially to compare speed and quality of the results.

6. Conclusions

The tool is very useful for both students and teachers involved in an implementation-oriented digital image processing course. Students find themselves free of image display and user interface tasks and can therefore concentrate on the image processing aspects of the implementation. Teachers have an environment which can be adapted for the particular characteristics of his course, and used to ease the task of evaluating the students’ work.

Future additions to the system include the use of more X-Window features to get user input, such as dials, radio buttons, and others, as well as the addition of other capabilities such as colormap editing and pseudocoloring.

References


