

Creating a Python GUI for a C++ Image Processing Library

Shaun N Wuth

Raymond Coetzee

Stephen P Levitt

School of Electrical and Information Engineering
University of the Witwatersrand
Johannesburg, South Africa

Abstract—The objective of this work was to create a extensible graphical user interface (GUI) using Python and Tkinter for an image processing library, IPL98, which is written in C/C++. The decision was taken to use Python in order to gain the advantages of rapid development associated with using a scripting language. Unfortunately, the productivity gains that were made by using simple Python scripting for generating the GUI (rather than a more complex C++ framework) were offset by the difficulties encountered when trying to generate Python bindings for the IPL98 library. The solution adopted was to wrap individual C++ functions packaging chunks of IPL98 functionality for importing into Python. This solution is sub-optimal in that there is too much of a division between the user interface and the application logic with the result that the GUI is unable to access image data that is in memory. To resolve this problem, future work will focus on wrapping the IPL98 classes directly concerned with image storage.

I. INTRODUCTION

Image Processing Library 98 (IPL98) [1] is an open-source image processing library. It supports functions from simple image scaling and rotating, to fourier transforms and feature extraction. IPL98 is coded in ANSI/ISO standard C++ and C and is generally compilable on platforms that have compilers which conform to the standard. The library consists of a kernel which is written in C and provides fundamental routines, and C++ classes which wrap and expand the kernel functionality. There are two broad categories of classes — containers and algorithms. Containers are used for storing images and algorithms are used for manipulating images. The contributors and designers of IPL98 have deliberately avoided making GUIs part of the library in order to remain platform independent.

Several unofficial add-on GUIs do exist but they are based on proprietary frameworks/toolkits which are restricted to the Windows platform [2]. The objective of this work was to rapidly develop an extensible cross-platform GUI for IPL98. The intention was also that the GUI could be easily modifiable without requiring the programmer to understand and master a complex toolkit.

The system that was developed is called GNU-Vu. The next section motivates the decision to use a scripting language for the GUI. Section III describes how the interface between the Python GUI and C++ library was constructed. This is followed by a discussion on the design of the GUI itself and the issues

encountered (section IV). Conclusions and recommendations regarding GNU-Vu complete the paper.

II. USING A SCRIPTING LANGUAGE FOR THE GUI

Scripting languages allow for rapid development and prototyping due to the following:

- They can be used interactively and do not need to be compiled.
- The run-time environment assumes the responsibility of memory management (i.e garbage collection) rather than having programmer take on this responsibility.
- They are dynamically typed as opposed to system languages like C++ which require the type of each variable to be explicitly stated. This allows the programmer to be less concerned with types when coding but does require more extensive unit testing in order to detect type mismatches occurring at run-time.

All in all, it is arguable that scripting languages are far easier to learn and use than traditional system languages [3]. Additionally, scripting languages are particularly useful as “glue” languages, in this case, gluing a user interface to application logic. Scripting languages are also interpreted, not compiled, and since the plain text source files can be directly parsable on a variety of platforms, it provides an ideal way for implementing a cross-platform application.

Note that various cross-platform C++ GUI toolkits exist [4] and the GUI could have been coded directly in C++; however, the advantages pertaining to rapid development would be negated.

The scripting language that was chosen was Python [5], due to the fact that it is well respected, portable, thoroughly documented, and has a clearly defined syntax and usage style. Furthermore, Python has bindings to various libraries including *Tkinter* [6], which provides an interface to the Tk GUI toolkit [7], and adds GUI design functionality to the language. Tkinter is in fact Python’s “de-facto standard GUI package” [6]. It offers an extensive widget set, so the most common and familiar elements of a GUI are available for all platforms.

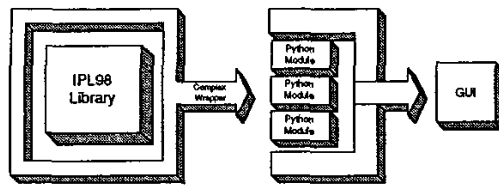


Fig. 1. Wrapping the Entire IPL98 Library

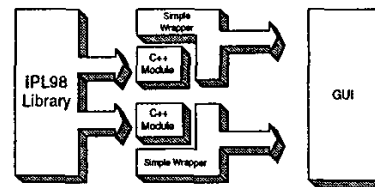


Fig. 2. Wrapping C++ modules

III. INTERFACING C++ AND PYTHON USING SWIG

The task of interfacing or binding Python with C++ libraries can be accomplished by using a tool called SWIG (Simplified Wrapper and Interface Generator) [8]. SWIG is capable of connecting C++ programmes with a variety of high-level programming languages, including Python [9].

SWIG wraps C++ code by processing a manually constructed interface file which contains function declarations, type definitions and other type information. When SWIG wraps, for example, a C++ function, two additional files are generated: first an interface to the source language (C++), and second, an interface to the destination language (Python). The generated C++ file has to be compiled, with the original C++ function, to produce a dynamically linked library file. This library file, together with the SWIG generated Python file, are the necessary components for importing the functionality into the destination language. SWIG is also capable of wrapping C++ classes by producing proxy Python classes.

A. Wrapping the entire IPL98 Library

The initial plan was to generate Python bindings for the entire IPL98 library. Once the library was wrapped and imported, it could be used natively from within the "easier" Python environment. This would also allow the GUI, which was being created using Python, to directly access IPL98's containers and algorithms. Figure 1 illustrates this wrapping approach — the entire IPL98 library is wrapped and used natively within Python modules. These modules package chunks of functionality which can then be accessed via the GUI. A chunk of functionality could be, for example, "cropping an image". In other words, each module is used to implement a use case (barring the GUI component). Within the "cropping an image" module code exists for instantiating an image object for the source image. The source image is loaded into memory and its dimensions determined. If the cropping boundaries are feasible then a destination image object for containing the cropped image is constructed and the appropriate pixels are copied from the source image to the destination image.

The binding process soon became cumbersome and unviable. The C++ code in the IPL98 library is fairly complex as it wraps an underlying C kernel and makes extensive use of templates, pointers and various data structures. All of these factors increase the complexity of the SWIG interface files which the programmer has to manually generate.

SWIG is capable of producing a proxy class in Python for each C++ class. However, a direct mapping cannot exist from C++ to Python because of the languages dissimilar object models, data structures, native types and so on. This will force the Python version of the library to be different (in some cases subtly) to the C++ version. This is undesirable as it is confusing to users and will require additional documentation describing the Python version. It is also worth noting that memory management becomes more complicated when instantiating objects and using pointers across the two languages.

All of the above reasons led to the decision to abandon the idea of wrapping the entire library.

B. Wrapping C++ Functions Performing Specific Tasks

The manner in which the above problems were resolved was to shift the implementation from Python to C++. Individual C++ functions were written making use of IPL98 library and offering chunks of functionality useful to an end user, such as the image cropping (refer to section III-A for more detail on this example).

These C++ functions use IPL98 natively, receiving and returning arguments to the GUI via the SWIG generated wrapper. Each function's parameters are constrained to primitive or enumerated types. In this way, the complexity of the IPL98 library (namespaces, template classes, class hierarchies, etc) can be hidden from SWIG (and the GUI).

C. Modular Development

The wrapping functions were made modular by virtue of the fact that only one function could be put into each module (more than one could be put in, if it was overloaded). This restriction was due to the strict naming and directory convention that is necessary to maintain the extensible structure of the project (section IV-B). Each module is imported separately and is completely self contained - it may be used in a C++ main function and executed in C++ (pre-wrapping), or be called from a simple Python programme, without the other modules or the GUI present. This is depicted in Figure 2.

The advantages of this approach are:

- The SWIG interface file for a single function is a great deal simpler than that for a large class. This means that the overall wrapping process is simplified.
- Memory problems cease to exist as the objects and variables constructed within each module function are

destroyed when the function has finished executing and they go out of scope.

- Modules are independent of one another. This allows modules to be modified without affecting any other modules.

Wrapping the functions as described above is not ideal, and there are a few drawbacks which need to be highlighted:

- The process of adding functional modules has to start in C++. If a programmer wants to add a new functionality, the module has to be written in C++ and then wrapped, as opposed to making a module from pre-wrapped classes and functions.
- There is no direct access to the image objects of IPL98 from within the GUI code. IPL98 image objects exist only for the duration of the wrapper functions' execution. If a user opens an image and proceeds to apply multiple algorithms to it and these algorithms happen to be distributed among different modules, then the image will need to be loaded from disk at the start of each module and saved to disk at the module's termination which is, of course, inefficient.
- Another consequence of the fact that there is no direct access to the image objects from within the GUI is that any functionality involving user interactions with an image, such as pixel selection using a mouse, is made much more difficult to implement.
- Although the IPL98 library functionality is limited, the number of combinations of IPL98 functions that could be made and wrapped is infinite. It is impossible to anticipate all the combinations of functions that could be executed. A few of the more common ones have been wrapped, but the onus is on the user of IPL98 to write new functions and wrap them if the functionality that is desired is not available. Note, that GNU-Vu does provide the structure to do this easily (section IV-B).

IV. THE PYTHON GUI

A. Requirements Specification

The primary requirement of GNU-Vu was that it was to be a GUI for IPL98, and not a graphics package similar to *Adobe Photoshop* [10], or the like. As a result it was decided that the target user of GNU-Vu would be one who knows how to use IPL98 already, and is simply looking for a simpler way to access its functionality. It would not be aimed at a novice image manipulation user, and thus a user of GNU-Vu would have to know the intricacies of IPL98, and when and how to use a chosen function.

It was decided that the GUI should be able to open, save, and display an image, as well as perform one or more of the IPL98 functions and algorithms on it. An additional, important requirement was that the GUI should also be extensible in that new functionality should be able to be included without redesigning the GUI. This is due to the fact that the IPL98 libraries are subject to ongoing development and are continually being revised and updated with new algorithms.

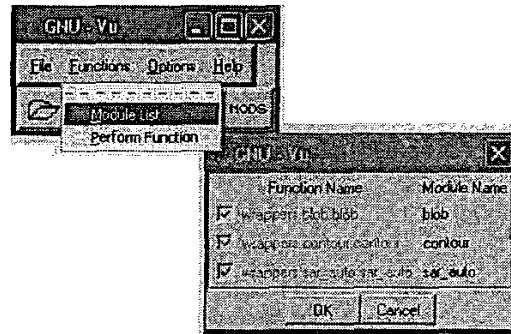


Fig. 3. The Module List Dialogue

B. An Extensible Interface

When the GUI is loaded, it scans a *wrappers* directory for modules. Each module forms a sub-directory and must contain the Python *import* file (automatically generated by SWIG), the *module description* file, and the compiled library file. The Python import file should have the same name as the sub-directory - this is how the GUI knows which file to import. The module description file contains the description of the wrapped function and each of its parameters. This is used by the GUI to describe the function to the user and to allow the user to specify arguments for the function.

If the directory structure described is adhered to and all the relevant files are present then each directory is treated as a Python package. The "Module List" dialogue (Figure 3) allows the user to choose which modules to import. On completion of the dialogue the imported functionality is immediately available. This dialogue is automatically updated when new modules are added to the directory structure, making GNU-Vu easily extensible. It is worth noting that the functionality that can be added to GNU-Vu is not limited to that within the IPL98 library. Any C++ function can be wrapped and accessed via the GUI, as long as it conforms to the module approach that has been adopted.

Another automated process in that of menu creation. Every menu shown in GNU-Vu is listed in a file. When GNU-Vu is run, this file is parsed, and the menus are created from the listed items. This means that the menu structure can be automatically updated every time a new module is added to the system.

C. Working Directory

A final mention of the *working* directory must be made. This directory is used to store temporary files generated by the IPL98 algorithms. When an algorithm is run it produces a temporary file which stores the image output. The content of the output file is displayed to the user via the GUI (Figure 4). This output file is deleted when the user closes the image, unless the user decides to save the image. This is a consequence of fact that images cannot be stored in-memory

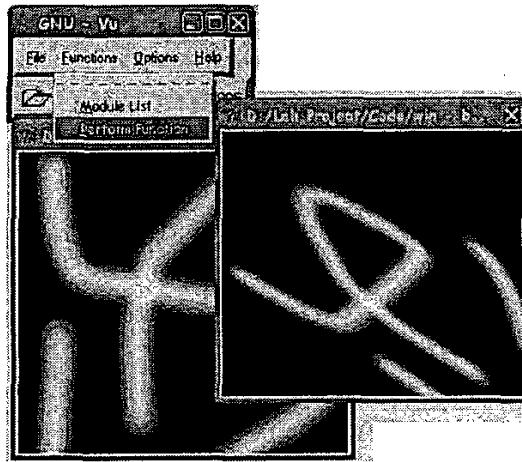


Fig. 4. Original Image and Scaled and Rotated Image

as the GUI does not have access to the IPL98 container classes.

D. Image Display

Displaying images using the GUI is an important requirement for GNU-Vu. This turned out to be more problematic than expected. IPL98 does not contain any functions for image display (see section I) so this meant that a display function in Python had to be used. Python itself also has no feature for this; and the Tkinter library does not support IPL98's primary format (BMP files). So, another library - the Python Imaging Library [11] - had to be used in conjunction with Tkinter. The bitmap first had to be converted to an intermediate format, and then displayed. Fortunately this did not degrade the source or destination images in any way.

V. CONCLUSIONS AND RECOMMENDATIONS

GNU-Vu is a cross-platform image processing application built from a mixture of programming languages: Tk, a scripting language for generating the GUI; C++, a system language which implements the image processing routines; and Python, a scripting language used to glue the GUI and the image processing library together. The motivation for using a scripting language for the GUI was to allow for rapid development and to obviate the need to learn a more complicated C++ toolkit or framework. Python was sensible choice as a glue language as it already possesses bindings for Tk in the form of Tkinter, and SWIG can be used to generate bindings for C++ code.

The use of Tkinter, did indeed, greatly ease the development of the GUI and will allow future programmers to understand and adapt the GUI without much of a learning curve. Unfortunately, the gains that were made in the arena of GUI were offset by the difficulties encountered in trying to wrap the IPL98 library. The final solution adopted was to create C++ functions packaging IPL98 functionality. These functions are simply wrapped and imported into Python.

This solution is non-optimal in that the presentation layer (GUI) is too decoupled from the application logic (IPL98) and there is currently no way in which images can exist in-memory beyond the lifetime of a module's execution. This means that IPL98 image objects are not directly accessible via the GUI which results in serious inefficiencies in certain usage scenarios (section III-C).

A resolution to these problems would be to wrap IPL98's image container class hierarchy — not the entire library. An investigation into the feasibility of doing this needs to be undertaken. If it can be achieved then using the corresponding Python proxy container classes in conjunction with the wrapping strategy currently employed will allow the GUI to send and receive image objects from wrapped C++ functions.

GNU-Vu demonstrates that Python and C++ are able to successfully interoperate but that it is not all "plain sailing". The difficulties encountered will presumably vary depending on the nature of the C++ libraries that are being wrapped and on the particular application being developed. It is worth noting that when wrapping a large library careful consideration needs to be given as to exactly which are the essential parts that need to be wrapped in order to allow the library to be used effectively.

REFERENCES

- [1] R. D. Eriksen. (2003) Image Processing Library 98. [Online]. Available: <http://www.mip.sdu.dk/ipl98/>
- [2] —. (2003) Unofficial IPL98 exchange. [Online]. Available: http://www.mip.sdu.dk/ipl98/unofficial/unofficial_ip98_exchange.htm
- [3] J. K. Ousterhout, "Scripting: Higher-level programming for the 21st century," *IEEE Computer*, vol. 31, no. 3, pp. 23-30, 1998. [Online]. Available: <http://citeseer.nj.nec.com/ousterhout97scripting.html>
- [4] L. Polzer. (2003) GUI toolkits for the X window system. [Online]. Available: <http://freshmeat.net/articles/view/928/>
- [5] (2004) Python. Python Software Foundation. [Online]. Available: <http://www.python.org/>
- [6] (2003) Python Tkinter resources. Python Software Foundation. [Online]. Available: <http://www.python.org/topics/tkinter/>
- [7] (2004) Tcl developer site. [Online]. Available: <http://www.tcl.tk/>
- [8] (2004) Simplified Wrapper and Interface Generator. [Online]. Available: <http://www.swig.org/>
- [9] (2003) SWIG and Python. [Online]. Available: <http://www.swig.org/Doc1.3/Python.html>
- [10] (2004) Adobe Photoshop. Adobe Systems Incorporated. [Online]. Available: <http://www.adobe.com/products/photoshop/main.html>
- [11] (2003) Python Imaging Library (PIL). [Online]. Available: <http://www.pythonware.com/products/pil/>