

Programming at the End of the Learning Curve: Lisp Scripting for Image Processing

Steven L. Tanimoto and Jeremy W. Baer

University of Washington

tanimoto@cs.washington.edu, jbaer@cs.washington.edu

Abstract

An image processing environment for students of information technology is presented in which programming is an ultimate objective. Our system offers a gentle entry into programming, and we argue that the manner in which programming is introduced is more important for learners than the specific syntax of the language. The environment and its student activities expose successively more sophisticated functionality that culminates in scripting using Lisp.

1. Introduction

The design of high-quality computer-based learning environments poses numerous challenges. These include the design of construction sets with primitives and combining rules, the design of tools and languages, and, in some cases, the careful integration of programming facilities. As part of the NSF-funded project on Mathematics Experiences Through Image Processing [1], we have experimented with the coupling of an image processing system to several programming languages. In our experience, the most effective combination has used Lisp as the programming language.

Here we explain some of the features of the combined environment and make several arguments in favor of scripting languages in general and Lisp in particular. An important part of our argument is that the particular syntax of the chosen language is not very important in comparison with the teaching context and approach used to introduce the language.

2. The METIP Environment

The objective of the project on Mathematics Experiences Through Image Processing is to enhance students' engagement with mathematics through the development and distribution of software and activities based upon image processing. The project has developed a succession of software systems beginning with the monochrome Pixel Calculator and culminating in Color

XFORM, which is an environment for viewing and manipulating color images using mathematical formulas. There are both PC and Macintosh versions available free of charge for academic purposes from the METIP project web site [1].

Color XFORM consists of the following components: an image display facility that clearly shows the RGB values and coordinate values for pixels; a means to apply mathematical operations to selected pixels via a calculator-like interface; an image transformation engine that can efficiently apply arbitrary formulas to images; and a means to program a sequence of operations that may involve transforming images as well as reconfiguring the layout of image windows on the screen.

At an early point in the project development, a need for a programming interface was identified. This was so that the image processing environment could be easily customized for particular activities, such as "Pixel Bingo", and demonstrating small-kernel convolution, etc. Later, an additional motivation for coupling programming environments with the image processing tools was so that we could teach programming in a way that takes advantage of the students' experience with the image processing tools.

The need for a programming interface led initially to a general solution and several experiments.

3. The Choice of Scripting Language

The METIP Programming Environment began as a language-independent module that could be coupled with any Windows-based language using Microsoft's Dynamic Data Exchange (DDE). This permitted using the image processing software with C, C++, Visual Basic, and XLISP-STAT. We implemented specialized versions of the software for convolution and stereogram making, making use of C++ and Visual Basic, respectively. However, the most interesting combination for us was using Lisp, because of Lisp's interactive nature.

The key factors influencing a choice of scripting language are the following: (1) expressiveness [2], (2) suitability for interpreter-mode, (3) learnability, (4)

expected return on investment, and (5) existing support for the language and for learning it (language maturity).

In the case of Lisp, one can make good arguments on each of these dimensions. While Lisp's syntax is unusual, possibly making it more difficult to learn, its interactive nature, properly exploited, makes it easier to learn. The return on investment for any scripting language includes both short-term returns, which we can identify with the use of the language within the image processing application, and long-term returns (uses in other contexts than image processing). Lisp can be used by itself or in other applications such as GNU Emacs and AutoCAD.

A custom scripting language, no matter how easy to learn it might be, does not have the property that it can be used in contexts other than the surrounding application.

As a scripting language for Color XFORM, the Lisp functionality divides into two parts: standard Lisp operators and environment-specific operators. The standard operators, such as arithmetic functions, list-manipulation functions, and facilities for control and function definition, are traditional and covered in numerous texts and online tutorials. On the other hand, the special functions for interacting with the image environment are not part of standard ANSI Lisp.

The image-processing specific functionality is encapsulated in an API (Application Programming Interface) that has an important property: it's *replay-complete*, which means that any sequence of operations a user can perform directly in the environment can also be performed using a sequence of Lisp calls to the API.

4. The Learning Curve

End-user programming typically means programming by users who do not know how to program before they get involved with a particular application program. Also, we don't expect them to program complicated procedures.

Whatever difficulties the programming language may pose we help the users to overcome by preparing them well in the context of the application and then taking them gradually through the programming constructs they need, relating each new feature to the image processing context or activity. The learning curve in our environment is gradual, because students get familiar with many of the computational operations on images via direct manipulation before they get to programming at all. When they do reach the programming stage, they want to do it in order to automate specific operation sequences that they have had to perform by hand.

5. Lessons Learned

Here are some of the lessons we have learned: (1) Don't count on the scripting language to handle highly

intensive computations itself; because of its interpretation, the Lisp execution is relatively slow compared with the image processing software. (2) It's expensive to provide a full IDE for the scripting language but not really necessary, because the intended uses are for small programs (scripts) that tend not to need sophisticated debugging tools. (3) Extra checks on resources are needed. For example, without scripting, users would have a hard time allocating 100 image buffers, but with scripting, it's easier, and therefore more important that there be error handling code for such things.

6. Discussion

Some comparisons can be made to other interactive environments for mathematics or image processing in which scripting is important. Mathematica is a custom programming language for a broad application area, but it is proprietary, and there is no gentle ramp up to the language through direct manipulation of graphical objects.

The ImageJ system by Rasband is a Java program for image processing that includes a facility for customization through Java-encoded modules. Those who would do scripting must write and compile Java code that can then be loaded into the running applet or application. While a user can perform direct manipulation of images and then write Java code, there is a big leap between them, and there are constraints on the code that make it awkward for novices to program in this context. Logo and Boxer are general programming systems for novices that can be customized to do image processing. While these systems are valuable learning tools, their direct-manipulation facilities for images are more limited.

7. Acknowledgements

The authors would like to thank the many developers of the Pixel Calculator and METIP Programming Environment and especially: Dennis Lee, James Ahrens, David Simmons, Alex Rothenberg, Evan McLain, Michael Van Hilst, and Lauren Bricker. Partial support from the National Science Foundation under grant MDR-9155709 is gratefully acknowledged.

8. References

- [1] Mathematics Experiences Through Image Processing web site. <http://www.cs.washington.edu/research/metip/>.
- [2] J.K. Ousterhout, Scripting: "Higher Level Programming for the 21st Century", *IEEE Computer*, March 1998.
- [3] S.L. Tanimoto, "Exploring mathematics with image processing", *Proc. 1995 IFIP World Conference on Computers in Education*, Chapman & Hall, London, pp.805-814.
- [4] S.L. Tanimoto, "Image processing and teaching the concept of function", *Proc. ED-MEDIA 2001*, Tampere, Finland.