LEARNING GRAPHICS PROGRAMMING BY DIRECT COMMUNICATION

Martin Tuori
Tim Pointing
Defence and Civil Institute of Environmental Medicine
PO Box 2000
Downsview, Ontario, M3M 3B9

ABSTRACT

The process of learning the graphics functions of a computer graphics workstation environment is both assisted and hampered by the presence of an intermediary programming language. Assistance comes in the form of programming language functions for preprocessing, storage declaration, expression evaluation, control flow, and system libraries. Working against the student, compilation of programmed examples is slow, and errors may arise both from the syntax and semantics of the graphics functions, and from those of the programming language.

We propose an approach to learning the graphics functions that temporarily separates the graphics component from other aspects of the overall programming environment; in a sense, we are proposing training wheels for the graphics subsystem.

This approach was used in creating, for the IRIS series of workstations, [1] a graphics interpretter that allows a student to test out graphics concepts, without the need to write programs. Subroutine calls typed to the interpreter are carried out immediately, allowing a quick, trial-and-error approach. We argue that this approach is a useful addition to conventional learning techniques, and that its success can be attributed to bringing the student programmer into more direct communication with the graphical components of the programming environment.

INTRODUCTION

A student learning the details of a new computer graphics programming environment may employ many different techniques. He may begin by reading the manufacturer's documentation, which, if well written, conveys basic concepts, syntax, semantics and suggestions for efficient use of the computer graphics system. While this is an important stage in the student's training, it is not enough to give him fluency as a graphics programmer. Writing small test programs is good way to proceed, and is made much easier if a sample skeleton program, or stub, is provided. As Duff says, "Whenever possible, steal code." [Duff 1985]. The student can extend the stub to exercise individual features of the graphics environment, or combinations of features, thereby gaining familiarity with the concepts and behaviour of the system.

A high-level programming language provides a variety of features that can help the student in his exploration of a system and its functions. Macro preprocessing serves two useful functions — common constants and expressions are provided in system files, for inclusion in new programs, and the student can define macros to suit his own needs. Storage declaration provides for complex object definition, and for loading them from external sources. Expression evaluation allows results from one operation to be used as input to another; for example, reading pixel values from a raster display, modifying and redisplaying them. Features for control flow allow conditional, iterative and recursive action. Finally, various support libraries for mathematical, input/output, networking and other functions offer specific functionality, as needed. Although the student can defer the use of some language features, such as specialized subroutine libraries, he cannot avoid the basic syntax and semantics of the programming language itself.

1. IRIS is a trademark of Silicon Graphics Inc.