Pinc: A Tool for Maintaining Configurable Software in Pascal*

*T.A. Cargill, M.W. Herman***

Department of Computer Science,
University of Waterloo,
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

The problems of structuring and maintaining a substantial, portable, configurable, device-independent computer graphics package in Pascal provide the motivation for an ongoing project in source text control. Solutions, adapted from techniques developed for managing the source of the Thoth operating system[3], are described. The means by which a general purpose file system is employed for source text management is described, together with a software tool Pinc (the Pascal Inclusion Builder) which selects a subset of the body of source for compilation of an instance of the graphics package. The proposed technique contrasts well with recent proposals for sophisticated software development environments[2,9], since it yields valuable support inexpensively and immediately in a conventional computing environment. An evaluation of the current implementation is presented.

Introduction

A computer graphics package[1], developed over the last three years by J.C. Beatty and K.S. Booth, is a major software vehicle for teaching and research in computer graphics in the Department of Computer Science at the University of Waterloo. It has also been used for research in graphics techniques at Tektronix Inc. The package supports a variety of output devices: Tektronix 4010, Tektronix 4027, Hewlett-Packard 2648A, Varian electrostatic plotter, Telidon videotex terminals, line printers and cursor addressed alphanumeric terminals.

The software is structured as a hierarchy of "levels", each of which receives requests from the level above and sends further requests to the level below. The five levels are:

Level 4 - Transformations, text, windowing, viewporting

Level 3 - Clipping

Level 2 - Normalized to virtual co-orindate mapping

Level 1 - Virtual to device co-ordinate mapping and device-dependent drivers

Level 0 - I/O primitives and disassemblers for device-specific codes generated by level 1

With the exception of a small number of bit manipulation procedures (which should be eliminated), the package is written entirely in Pascal. Compelling pedagogical arguments favored Pascal as the implementation language and the decision appears to be sound[1].

The native environment of the package is TSS timesharing under the GCOS8 operating system on a Honeywell 66/60. The Pascal processor is Pascal/66. The graphics package has also been used extensively under the Unix operating system on PDP-11's and will be ported to a VAX/780 this year. It will also be used for bit-mapped graphics on a microcomputer personal workstation in development.

*Research supported by the Natural Sciences and Engineering Research Council of Canada under grants A5046, A4309 and A3022 and the University of Waterloo.

**Present address: Dome Petroleum, Calgary, Alberta

Motivation

The graphics package is *not* a program in the Pascal sense. That is, it is not a monolithic Pascal block which is to be compiled and run. Rather, it is a collection of related components which can be configured and variously combined with applications drivers to form a *family* of graphics systems. The members of this family vary in

- the "level" at which the application program interfaces with the package
- the device(s) supported by the package
- the sophistication of some graphics algorithms (e.g. Bezier patches)
- the inclusion or exclusion of such algorithms (e.g. visible surface processing)
- the functional completeness (when used as the basis of student exercises)

In addition, more variation is anticipated as the package is used in more environments. While variants needed to accommodate non-standard compilers will probably be few, that cannot be said for variants required for interfacing to idiosyncratic operating systems. Anticipating the need for variants and designing in such a way that they can be incorporated with moderate ease is crucial to the success of the package.

Though it is possible to do so, rarely is an application driver combined with the graphics package source as a monolithic Pascal compilation. Normally each "level" of the package (as indicated above) is compiled separately. The application driver is then linked with the appropriate libraries by a conventional linkage editor (which does not have any special support for Pascal). While universally referenced Pascal **const** and **type** declarations are used in the compilation of all procedures, there is no type-checking on the communication between procedures of distinct levels when they are compiled into separate libraries, level by level. The use of separate compilation is mandated by the prohibitive cost of the monolithic compilation of approximately 5,000 lines of Pascal in timesharing.

A means of managing this body of Pascal source is called for so that the type of operations indicated above can be performed with relative ease while the package as a whole undergoes normal maintenance and evolution. The package is modified regularly to enhance its function and to add new devices and algorithms (in addition to the correction of errors, of course). While only a small number of programmers (faculty, research assistants, teaching assistants and students) alter the package at a given time, the set of programmers involved changes frequently with student turnover.