POSTSCRIPT™

Paul Snow, Cliff Click, and Norman Naugle
P.O. Box 10162
College Station, Texas 77840

July 9, 1986

Abstract

POSTSCRIPT is a page description language developed and implemented by Adobe. Adobe appears to have implemented POSTSCRIPT in C, but we thought **Forth** would be better suited for the job. We explored the issue by developing X–SCRIPT in a **Forth**-like programming environment called **Fifth**. We soon learned much of POSTSCRIPT is designed to fit in nicely with common C functions. Much of **Forth**'s flexibility and speed is hampered by POSTSCRIPT's inflexible design.

Introduction

POSTSCRIPT[1] is page formatting language, a standard that has been proposed by Adobe systems, Inc. It uses a postfix notation very similar to **Forth**. POSTSCRIPT can be easily generated by other programs, or by programmers, and can produce stunning graphics and text. POSTSCRIPT's fonts are of literally *any* size or slant. Many fonts are outline fonts, and can be used to produce stunning effects when manipulated as graphics.

The motivation behind POSTSCRIPT is to take advantage of raster output devices like laser printers. These printers far more versatile than line printers, but their capabilities are neglected by outdated printer communication conventions. POSTSCRIPT is one scheme for accessing these capabilities.

We were four months into developing **Fifth**, a **Forth**-based programming environment, when Norman Naugle approached us with the idea of implementing X–SCRIPT, a POSTSCRIPT look-alike. We were to implement X–SCRIPT in **Fifth** on a PC class machine. The output devices for X–SCRIPT include various laser printers and graphics screens. We have the major pieces together and working, and hope to be 100% finished by the end of the summer.

POSTSCRIPT's Specifications

X–SCRIPT follows POSTSCRIPT's definition as given in the POSTSCRIPT manual put out by Adobe. The main challenge in implementing X–SCRIPT is that of developing a efficient system. POSTSCRIPT is not a particularly fast language. Originally we felt that tight **Forth** code and spot optimization would solve the speed problem. After doing some design and study, we came to the conclusion that the speed problem is not in the implementation of POSTSCRIPT. It is in the specifications. The POSTSCRIPT specifications clearly state:

- All routines do type checking.

[1]PostScript is a trademark of Adobe Systems, Inc.

- Many routines perform type conversions, if necessary.
- Many routines are overloaded.
- All functions are looked up at run time (Dynamic name binding).

X–SCRIPT uses a tagged stack to enable type checking and conversion. This tag allows X–SCRIPT to identify the type of any element in the system. Every X–SCRIPT element on the stack is made up of two **Fifth** elements. The extra **Fifth** element contains type information. There are 14 basic types in X–SCRIPT, with some types having special fields. (For example, strings and arrays have a length field.)

All primitives check operand types to determine if type conversions are necessary. If the primitive is overloaded[2] the check is necessary to decide what to do.

X–SCRIPT's dynamic name binding is the biggest slow down in the system. Procedures are lists of procedure names which are dynamically looked up at execution time. The process of looking up the name to find the procedure is called name binding. **Forth** does compile time name binding. (Procedures are a list of execution addresses.) In POSTSCRIPT name binding occurs at execution time. The results of the name binding is determined by something X–SCRIPT calls the dictionary stack. The dictionary stack specifies which dictionaries (an X–SCRIPT dictionary is very similar to a vocabulary in **Forth**) should be searched and in what order. If two definitions of procedure exist in different dictionaries on the stack, the definition in the dictionary highest in the dictionary stack is used. Changing the order of the dictionaries on the dictionary stack can alter dynamically the behavior of any or all procedures.

Unlike **Forth**, X–SCRIPT supports no "extensibility." Names are governed by rules typical of C, and certain characters are delimiters in X–SCRIPT's rigid syntax. This gives X–SCRIPT a familiar feel if one's background is in C, but limits **Forth**'s advantages as a implementation language.

Fifth and X–SCRIPT

A "standard" **Forth** system has several drawbacks as a starting point for an implementation of X–SCRIPT. **Forth** needs:

- Floating point.
- Graphics.
- Memory management.
- Large memory model.

Fifth has a large memory model, floating point numbers, simple graphics and a heap manager. Other capabilities of **Fifth** that turned out to be useful included a tree structured dictionary, dynamic compilation, and mutual visibility of modules. The tree structured dictionary is used to "hide" the complex inner workings of some of the modules. A module's name can be interpreted in the context within the dictionary; this reduces name collisions.

The dynamic compilation ability allows the program to change output devices by recompiling a set of device dependent modules. This avoids the run time overhead of choosing

[2]Overloading is where the same operator (or routine) can take parameters of various types. For example in C, the + symbol can mean integer addition or floating point addition, depending on the type of the expressions involved.

the output device by replacing it with a compile time decision. The decision can be easily changed by recompiling the device dependent modules.

The mutual visibility of modules is an ability granted by the tree structure. A module can "see" all its children, and all its children can "see" it. Recursive algorithms (and definitions) become easier to write.

Hash Tables

Dynamic name binding is powerful, but slow. We were able to increase the speed of dynamic name binding by using a self optimizing hash table. The main dictionary in X-SCRIPT, the system dictionary, has over 200 entries and is 95% full, making it a very dense hash table. Because the system dictionary table is so dense, almost all lookups in the table have a collision. At the point of the collision we begin a linear search for the hash key. Without any optimization, several key compares are done for each lookup. However when we find the key, we swap the key with the previous hash table entry. This moves the hit key closer to it's hash position, but moves another key further away. The long term effect of this behavior is that commonly used entries migrate toward their hash points. Uncommon entries are pushed about the table, but since they aren't commonly used no one cares.

Summary

POSTSCRIPT is designed to take advantage of the new high resolution printing devices, like laser printers. POSTSCRIPT is a flexible and convenient way to make outstanding page layouts. But the specifications from Adobe contain design decisions that limit its speed. The biggest speed problem is POSTSCRIPT's dynamic name binding. To speed up name binding, X-SCRIPT organizes dictionaries as self optimizing hash tables.

X-SCRIPT demonstrated the need for attention in certain areas of **Forth**. **Forth** needs a large memory model, floating point arithmetic, graphics, and memory management in order to implement systems like X-SCRIPT. It is our opinion that **Forth** needs some form of easy-to-use scoping. We certainly got a lot of mileage out of the tree structured hierarchy in **Fifth**. We hope some work is going in to extending the **Forth** standard; there is nothing we know of in the **Forth** philosophy that precludes the addition of those features **Forth** lacks.

On the bright side, X-SCRIPT has shown that **Forth** can be readily applied to large systems software. **Forth**'s immediate test and debug reduced development time and increased the reliability of X-SCRIPT. And when necessary features were lacking, we found **Forth** easy to modify. This is perhaps **Forth**'s biggest advantage over "traditional" languages; **Forth** *is* flexibility.

Yet To Go...

X-SCRIPT is in a workable form, and is nearing the alpha test stage. X-SCRIPT is competitive with POSTSCRIPT in speed, and with some planned enhancements not yet installed it could be considerably faster. X-SCRIPT currently outputs to the TI PC screen (720x300), the IBM PC screen (320x200, B&W or color), the QMS 800 laser printer, the Apple LaserWriter, and the JLASER card by TallTree.

X-SCRIPT currently lacks outline fonts (we use Hershey fonts), font bitmapping and caching, thick lines, and clipping.