A High Order Graphics Console

M.W.Blake-Knox
C.H.M.Griffiths
G.C.Morris
L.G.Woolsey

Bell-Northern Research
Ottawa, Canada

Abstract

The GRAPPLE Console is a single user interactive graphics system that executes the GRAPPLE language. It appears to the user to be a high order computer that is programmable in GRAPPLE. The Console consists of a mini-computer, a storage tube graphics display and a floppy disk. It is suitable for any work involving line drawings and computation. Several uses of the Console are examined from the point of view of their practicality and their economy.

Une Console Graphique de Rang Elevé

M.W.Blake-Knox
C.H.M.Griffiths
G.C.Morris
L.G.Woolsey

Recherches Bell-Northern
Ottawa, Canada

Abrégé

La console GRAPPLE est un système graphique à action réciproque pour un usager et qui exécute le langage GRAPPLE. Pour l'usager elle se présente sous la forme d'un ordinateur d'ordre supérieur qui est programmable en GRAPPLE. La console se compose d'un mini-ordinateur, d'un écran à présentation visuelle à memoire et d'un disque flexible. Elle est faite pour tout travail impliquant des dessins à lignes et des calculs. Plusieurs utilisations de la console sont envisagées du point de vue de leur aspect practique et économique.

* The authors wish to thank X.N.Dam for his assistance in coding the system and G.Scott for originating GRAPPLE.

Introduction.

By ignoring the myth that computer graphics must be by its very nature both expensive and cryptic, the authors at Bell-Northern Research, have designed and implemented a versatile low-cost computer graphics system that is easy to use. This system, the GRAPPLE Console, is a single user interactive graphics system that executes the GRAPPLE language. In this sense, it is considered to be a "high order machine". It consists of a 16 bit word mini-computer that drives a storage tube graphics display and uses a floppy disk [1] for auxiliary storage (fig. 1). The user enters GRAPPLE language statements through the display's keyboard. Through these statements, the user has complete control over the functions of the GRAPPLE Console. It is suitable for any work involving line drawings and computation. This might be the storage, retrieval and updating of drawings, or the simulation of processes with graphical output.

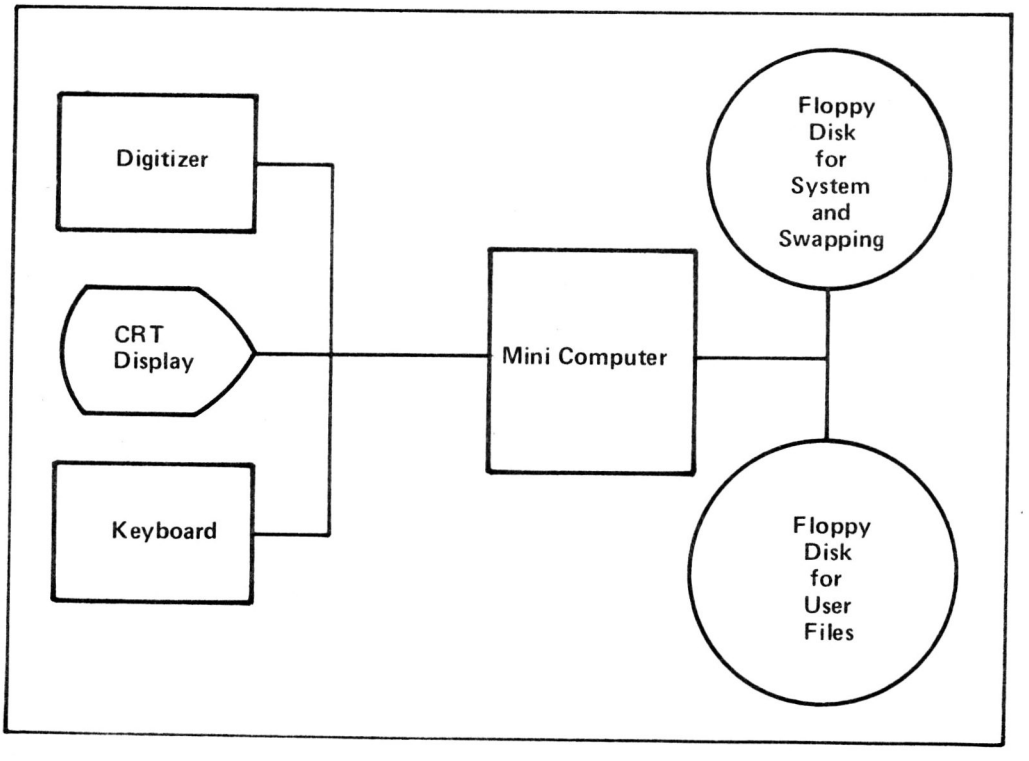

Fig. 1

This paper describes the philosophy behind GRAPPLE and the GRAPPLE Console. It describes and comments on our implementation techniques, before talking about the results obtained.

The GRAPPLE language.

The GRAPPLE language itself is a high order function oriented programming language, developed at Bell-Northern Research to permit the easy development of graphics applications [2,3,4,5]. It is now used in many design aids activities and has gained wide acceptance in our user community. GRAPPLE is available on our large host computer, an IBM System 370/168, which we operate in a time-sharing mode. It provides users with an excellent tool for graphic manipulation at storage tube terminals, at which they can readily interact in their editing processes.

GRAPPLE is a compiler-interpreter system. Source statements are compiled into an internal representation known as GRIT, or GRAPPLE interpreter text. GRIT is then interpreted to produce the desired graphic results. No distinction is made in GRAPPLE between data and program. A GRAPPLE program is both the program and the data on which the program operates. Functions defining data elements are handled identically to computational functions. GRAPPLE is a stack oriented language, a feature that enables function call nesting and recursion to an arbitrary depth. Stacks are used for parameter passing, execution sequencing and graphic environment control. GRAPPLE has developed from a simple picture description language to be a sophisticated graphic manipulation tool, in fact, a system for graphic work.

In the following brief example, code is shown to display an isosceles triangle in the middle of the display screen (fig. 2). The units are arbitrary, based though on a display screen size of 3800 units by 2900 units, with the origin in the bottom left hand corner.

```
TRIANGLE: V(800,0,-400,500,-400,-500);
S(1000,1000),*(2)TRIANGLE;
```

The statement defining TRIANGLE does so in terms of a relative vector with three segments. Each segment is specified in terms of the relative displacement of its end point from the current beam or pen position. The second statement is an imperative, asking for the primitive function S (Set to an absolute location on the screen) and the user defined function TRIANGLE to be executed. When TRIANGLE is invoked, the six coordinates are passed as parameters in the main stack to the relative vector processor V. This processor removes pairs of coordinates from the stack as it builds each vector. The second statement moves the drawing beam to the centre of the screen, and invokes TRIANGLE at twice its specified size. The '*' is a shorthand modifier, implying change of scale. The scale is changed by a factor of two, producing a triangle whose base is therefore 1600 units long.

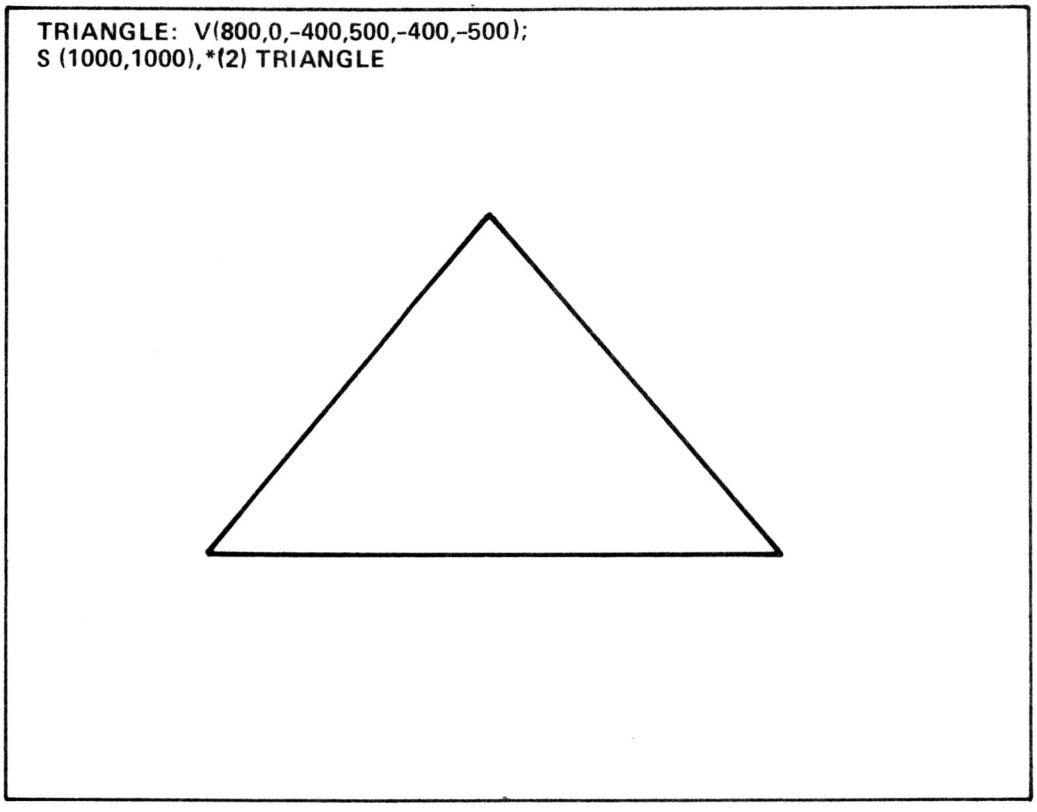

Fig. 2

This example shows two basic features of GRAPPLE: function calling and parameter passing. GRAPPLE statements may include graphic data, textual data and both logical and arithmetic expressions. As well as operations at a display screen (picture drawing and coordinate digitizing), GRAPPLE can access disk files to retrieve and create data. These features combine to make it a successful programming language.

Why a GRAPPLE Console?

The GRAPPLE language evolved to meet specific computer graphics requirements in our laboratories. The GRAPPLE Console evolved as a research project based on the language. The nature of GRAPPLE allows the following analogy. The intermediate code, GRIT, produced by the compiler, can be considered the machine language for a GRAPPLE machine [6]. The interpreter then embodies the CPU functions of that machine. The research project was conceived to examine the implications of a stand-alone GRAPPLE machine, since known as the GRAPPLE Console.

There were naturally other motivations. Ever since work was begun on GRAPPLE we had been concerned with transportability of GRAPPLE files between the several design aid tools then available to our designers. By making the software tools available across the board, the designer was free to use the most applicable (or available) tool. The GRAPPLE Console is a natural extension of these efforts, an attempt to provide a single user graphics system free of transportability problems.

Other reasons relate strongly to experience of our time-sharing system. Graphics traditionally demands faster response than the average use of computer facilities, and GRAPPLE is no exception. We recognized that a stand-alone system would be able to give exceptional response to graphic digitizing, and additionally provide a continuous digitizing mode not possible on the time shared system. Display output would also be several times faster, without the long breaks in tranmission due to system contention frequently experienced under time-sharing. There were also potential economies to be realised compared to a time sharing approach, as well as improved reliability and availability.

There were two main reasons for the development of the Console. Research into high-order machines interested us, as did continuing our exploration of GRAPPLE. The marriage of the two made a good research project. Also, the implications of distributed computing were being much discussed. The cost effectiveness of a time sharing approach to all computing was being seriously questioned, and this project would provide further fuel to the argument.

Logical Structure.

The GRAPPLE Console operates in much the same way as the GRAPPLE processor on our host computer (fig. 3). GRAPPLE language statements entered at the keyboard are compiled by the compiler, producing the internal code GRIT. If the statement is an execution imperative, then the interpreter is invoked to operate on the compiled GRIT. During compilation, access may be made to libraries of pre-defined GRAPPLE functions held on disk. During execution, other floppy disk files containing data may be referenced. In order to accomodate reasonable GRIT sizes, it is necessary to page the GRIT code in and out of main memory. Again the floppy disk is used to hold the paged data. As display functions are executed the results appear on the display screen.

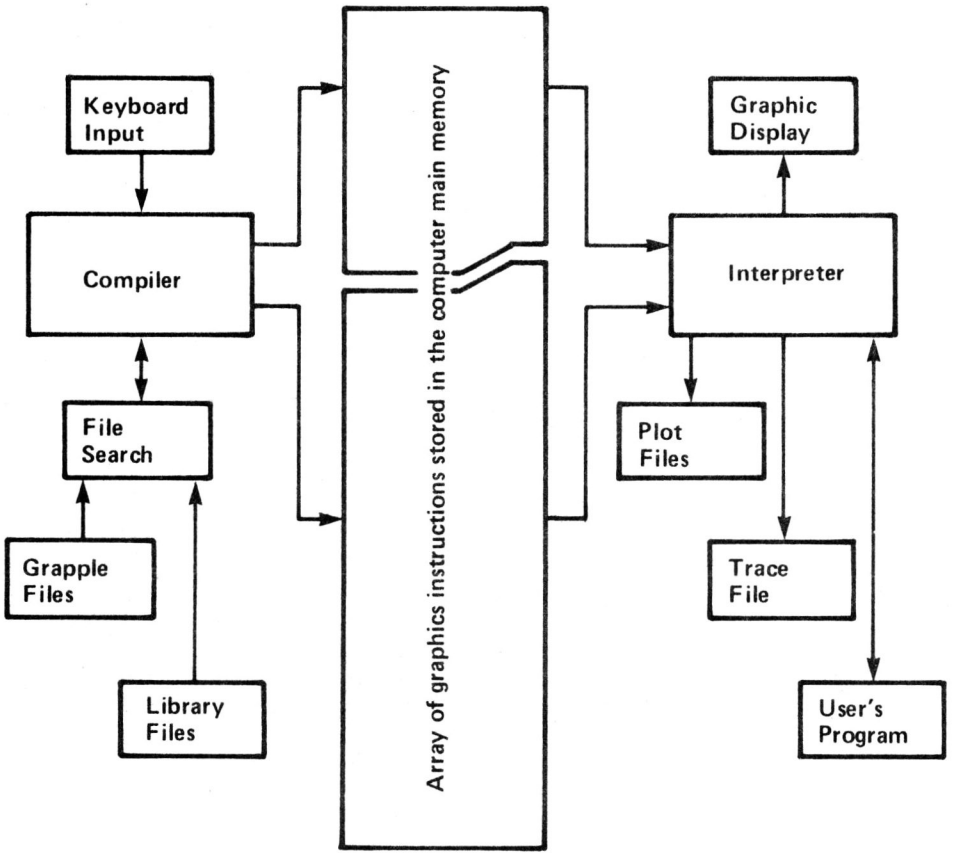

Fig. 3

There is a close association between the GRAPPLE source code and the internal GRIT code. GRAPPLE is basically a "direct language"; its compilation is a translation and reordering of one set of tokens for another. This feature makes compilation simple, yet it also allows decompilation (fig. 4). Thus it is possible to work backward from the internal GRIT format to produce a GRAPPLE source string. This is a very useful feature for graphic editors.

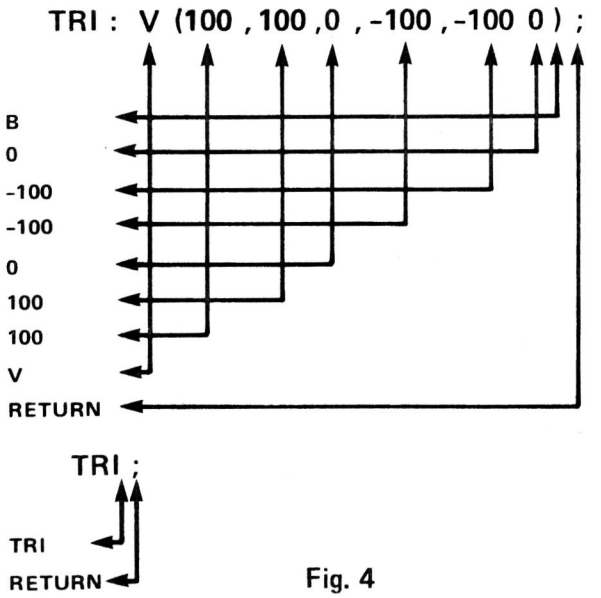

Fig. 4

Since the GRAPPLE language is very powerful and concise, and since it is also a direct language, the primitive operators are also very powerful. This implies that they operate for a "long time". Hence the access speed of the memory that holds these opcodes can be quite slow. For example, it requires three GRIT words to generate a vector. That vector would be described by a string of 11 characters sent to the display screen. Since those characters are sent at 9600 baud, the access time for each GRIT word in memory need only be 3.8 milliseconds. This is easily within the capabilities of a good paging scheme using a floppy disk, and a 1 microsecond processor.

The system is highly dependent on floating point, to the extent that the GRIT storage addressing is done in floating point. This at first seems cumbersome, but in a system where all numbers are real, it adds great generality.

Implementation.

Code for the GRAPPLE Console has been produced exclusively in assembly language. Speed of execution was of paramount importance, so that there was no question of the use of a high-level language like FORTRAN. In the early stages of the design work, we had hopes of using microcode. We found though, that the state of the art was not such that we could take advantage of it. Potentially we could have made the code more efficient, but it would not have been as intelligible as assembly code.

The system was constructed as a series of overlays with a resident section containing the basic service routines for I/O handling and overlay management. Overlays were required

since we had chosen to work with a 24K machine. Similarly in order to accomodate reasonable sized GRAPPLE programs, it was necessary to page the GRIT code to and from the floppy disk.

The compiler was based on a version written in XPL [7] using a BNF description of the syntax. This was prepared to provide a standard description of the GRAPPLE language for both the GRAPPLE Console, the host and any future implementations of GRAPPLE that may be done. The BNF syntax was reduced to SLR(1) form, using a syntax analyzer developed by DeRemer [8], modified to produce reduced matrix parser tables [9]. These tables in turn are the basis for the XPL GRAPPLE compiler and the compiler for the GRAPPLE Console.

The rest of the code was produced from "pseudo-code". "Pseudo-code" is a design technique whereby the program logic is designed in a semi-formal way [10,11]. Our pseudo-code was structured, and intended to be closely followed by the code when finally written. The language used to build the pseudo-code however is reasonably free English, allowing as clear as possible expression of the functions to be performed. By thinking through the program logic in a programming form, but before the code is actually cut, we ensured relatively bug-free code. Our experience of using this technique bears this out, for the system was running two days after we started integration of the code.

Both the interpreter overlay and the resident section were written from pseudo-code. The interpreter consists of a number of well-defined subroutines. The resident section consists of service routines and manufacturer's code. One aspect of the design was the reliance on floating point arithmetic. In order to give the greatest significance to the GRIT order codes, a special 32-bit floating point format was adopted. The floating point arithmetic routines were included in the resident section.

A second key aspect of the design was the algorithm to be used for GRIT page swapping. Analysis conducted of the way in which GRAPPLE operated on our host computer, gave us an idea of the optimum page size for several possible page replacement algorithms [12]. We experimented with several algorithms before adopting a 64 GRIT word page size and a simple Least Recently Used (LRU) replacement algorithm. This has worked out very well in practice.

The experience of implementing the GRAPPLE Console from our pseudo-code showed that we had minimized our debugging problems. Large parts of the code worked the first time because of the thorough work done in designing the pseudo-code. Two key problems we ran into were the level

of detail of the pseudo code, and real-time problems. The amount of detail required in pseudo-code turned out to be dependant on the difficulty of the module. As a result, some pseudo-code had to be reworked at coding time. However this did not represent a serious delay at the coding stage. The real time problems were more serious and produced most of the bugs during system integration. There was tremendous satisfaction though, in being able to bring the system up on schedule, November 1st 1974.

Results.

The first demonstration of the GRAPPLE Console's capabilities was to run a GRAPPLE program originally written for the host computer. Having done this once, we have since been able to show compatibility and transportability for a number of other host GRAPPLE applications. The following applications have been demonstrated on the prototype: Computer Aided Instruction, automatic programming, graphic editing, schematic displays and animation.

Valuable experience in the use of floppy disks has been gained. These have turned out to be a reliable media for data storage, albeit slow. We have been concerned with improving their performance. An indicator was placed on the shaft of the stepping motor, which rotates as the head seeks. Movement of this indicator exposed some very interesting consequences of our file structure. As a result some redesign took place.

The response of the system has more than met our design objectives. The storage tube display we have used has been interfaced at 9600 baud directly to the teletype interface of the mini-computer. This produces excellent interactive response at the display.

Finally the reliability of the system combines with the other features to produce a highly cost-effective system. Designers can operate at this kind of work station, unhindered by the vagaries of a central time-sharing system, with therefore enhanced effectiveness.

Conclusion.

Computer graphics has long been a field thought to be expensive and complex. The development of GRAPPLE and now the GRAPPLE Console at Bell-Northern Research, has shown that this need not always be the case. Graphic work involving dynamic three-dimensional rotation is very elegant and attractive, but rarely is it applicable to problems in our laboratories.

References.

1. S.Davis, Disk Storage for Minicomputer Applications, Computer Design, vol. 12, no. 6, June 1973, pp 55-56.

2. GRAPPLE Language Reference Manual, Version 4.0, Manual 13500, Bell-Northern Research, Ottawa, Canada.

3. D.L.Williams, GRAPPLE-Graphics Application Programming Language, Proc. 3rd Man-Computer Communications Seminar National Research Council, Ottawa, Canada, 1973.

4. R.B.Duncan, GRAPPLE Applications, Proc. 3rd Man-Computer Communications Seminar, 1973, pp 6.1-6.9.

5. G.Scott, C.H.M.Griffiths, R.B.Duncan and D.L.Williams, GRAPPLE - An Interactive Computer Graphics Language, TELESIS, vol. 3, no. 2, Summer 1973, pp 47-54.

6. L.G.Woolsey, Design for a High Level Graphics Language Machine, Bell-Northern Research, Ottawa, Canada, (To be published in INFOR, vol. 13, no. 3, October 1975).

7. W.M.Mckeeman, J.J.Horning and D.B.Wortman, A Compiler Generator, Prentice Hall Inc. 1970.

8. F.L.DeRemer, Simple LR(k) Grammars, CACM, vol. 14, no. 7, July 1971, pp 453-460.

9. M.L.Joliat, Translator Writing Systems at Bell-Northern Research, 8th Hawaii International Conference on System Science, January 7-9, 1975.

10. P.Bridges, S.Iscovici and B.Mitchell, Towards Quality Software, TELESIS, vol. 3, no. 8, July/August 1974, pp 232-238.

11. H.D.Mills, Chief Programmer Teams - Principles and Procedures, IBM, FSC 71-5108.

12. W.W.Chu and H.Opderbeck, Performance of Replacement Algorithms with Different Page Sizes, Computer, (IEEE), vol. 7, no. 11, November 1974, pp 14-21.

13. A.Van Dam and G.M.Stabler, Intelligent Satellites for Interactive Graphics, Proc. NCC 1973, pp 229-238.

14. T.H.Myer and I.E.Sutherland, On the Design of Display Processors, CACM, vol. 11, no. 6, June 1968, pp 410-414.

15. A.Hassitt, J.W.Lageshultz and L.E.Lyon, Implementation of a High Level Graphics Machine, CACM, vol. 16, no. 4, April 1973, pp 199-212.